МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

Кафедра інформаційних технологій

І.М. ЛАЗАРОВИЧ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни

«Комп'ютерні системи»

для студентів напряму підготовки
«Комп'ютерна інженерія»

ІВАНО-ФРАНКІВСЬК – 2014 РІК

УДК 004.2
ББК 32.973
Л17

Рекомендовано до друку вченою радою факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника, (протокол № 3 від 30 жовтня 2014 р.)

Рецензенти:

Філевич П.В. – доктор фіз.-мат.наук, професор, завідувач кафедри інформаційних технологій Прикарпатського національного університету імені Василя Стефаника;

Соломко А.В. – кандидат фізико-математичних наук, доцент кафедри математичного і функціонального аналізу факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника

Лазарович І. М.

Л17 Конспект лекцій з дисципліни «Комп'ютерні системи» для студентів напряму підготовки «Комп'ютерна інженерія» / І. М. Лазарович. – Івано-Франківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2014. – 190 с.

Подано класифікацію та наведено теоретичні основи організації та функціонування сучасних ком'ютерних систем, проаналізовано їх переваги і недоліки та сферу застосування. Розглянуто статичні та динамічні топології сучасних багатопроцесорних систем, проаналізовано способи маршрутизації та алгоритми вибору топологій при проектуванні структури комп'ютерної системи.

Наведено відомості щодо освоєння способів реалізації алгоритмічного та програмного забезпечення на основі найпоширеніших методів та алгоритмів обчислень. Проаналізовано сучасні способи організації взаємодії процесора і пам'яті, методи забезпечення когерентності пам'яті. Наведено основні показники надійності обчислювальних систем та підходи з покращення цих показників. Розглянуто питання діагностики та експлуатації комп'ютерних систем

Конспект лекцій розроблено на основі вимог освітньо кваліфікаційної характеритстики та освітньо-професійної програми підготовки фахівців з напряму «Комп'ютерна інженерія»

УДК 004.2
ББК 32.973

ЗМІСТ

Передмова 2
ПЕРЕЛІК ПОЗНАЧЕНЬ І СКОРОЧЕНЬ 3
РОЗДІЛ 1. СТРУКТУРИ ПАРАЛЕЛЬНИХ І РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ (ПРКС) 5
Тема 1.1. Комп'ютерні системи та паралельна обробка інформації .. 5
1.1.1 Загальні принципи функціонування комп'ютерних систем ... 8
1.1.2 Класифікація КС 8
Тема 1.2. КС класу SIMD 12
1.2.1 Векторні і векторно-конвеєрні КС 12
1.2.2 Матричні обчислювальні системи 17
1.2.3 КС нетрадиційної архітектури 22
Тема 1.3. Теорія обчислювальних систем 33
1.3.1 Основні поняття ТОС 33
1.3.2 Задачі ідентифікації та синтезу комп'ютерних систем 34
Контрольні питання до розділу 1 40
РОЗДІЛ 2 СУЧАСНІ ОБЧИСЛЮВАЛЬНІ СИТЕМИ СИСТЕМИ ТА ЇХ ТОПОЛОГІЇ 41
Тема 2.1 Мультикомп'ютерні КС
2.1.1 Комп'ютерні системи класу MIMD 41
2.1.2 Архітектури SMP 42
2.1.3 Кластерні обчислювальні системи 50
2.1.4 Системи з масовою паралельною обробкою (MPP) 59
2.1.5 Обчислювальні системи з неоднорідним доступом до пам'яті 67
Тема 2.2 Топології обчислювальних систем 71
2.2.1 Основні поняття ОС 71
2.2.2 Методи опису характеристик мережених з'єднань 75
2.2.3 Статичні топології 78
2.2.4 Динамічні топології 87
2.2.5 Функції маршрутизації даних в динамічних топологіях 93
Тема 2.3. Відмовостійкі ПРКС 98
2.3.1 Основні поняття відмовостійкості 98
2.3.2 Забезпечення відмовостійкості дискової пам'яті 102
Контрольні питання до розділу 2 117
РОЗДІЛ 3. ОРГАНІЗАЦІЯ ОБЧИСЛЕНЬ В ПРКС 118
Тема 3.1. Операційні системи КС 118
3.1.1 Основні поняття операційних систем 118
3.1.2 Види операційних систем 122
Тема 3.2. Механізми взаємодії процесів 126
3.2.1 Основи взаємодії процесів 126

3.2.2 Особливості міжпроцесової взаємодії 128
3.2.3 Основи передавання повідомлень 130
3.2.4 Технології передавання повідомлень 133
Тема 3.3 Паралельні алгоритми 137
3.3.1 Основи паралельних алгоритмів 137
3.3.2 Рівні розпаралелювання 139
3.3.2 Паралельні операції 141
Контрольні питання до розділу 3 143
РОЗДІЛ 4. ОРГАНІЗАЦІЯ ПАМ'ЯТІ ТА ВВЕДЕННЯ-ВИВЕДЕННЯ В КОМП'ЮТЕРНИХ СИСТЕМАХ 144
Тема 4.1. Системи введення-виведення (СВВ) 144
4.1.1 Функціонування комп'ютерного введення-виведення 144
4.1.2 Функції та структури систем введення-виведення 147
4.1.3 Засоби суміщення операцій обробки та введення-виведення 152
Тема 4.2. Організація пам'яті в ПРКС 153
4.2.1 Загальні питання організація пам'яті 153
4.2.2 Моделі архітектури пам'яті обчислювальних систем 156
4.2.3 Моделі архітектур з розподіленою пам'яттю 161
4.2.4 Мультипроцесорна когерентність кеш-пам'яті 163
Контрольні питання до розділу 4 167
РОЗДІЛ 5. НАДІЙНІСТЬ ТА ЕКСПЛУАТАЦІЯ КС 168
Тема 5.1. Надійність КС 168
5.1.1 Основи надійності КС 168
5.1.2 Методи підвищення надійності та контролю КС 170
Тема 5.2. Експлуатація КС 176
5.2.1 Основи технічного обслуговування КС 176
5.2.2 Автономна система ТО 176
5.2.3 Централізована система ТО 177
5.2.4 Змішана система ТО 179
Тема 5.3. Діагостика КС 180
5.3.1 Основи діагностики КС 180
5.3.2 Методи виявлення збоїв чи відмов у КС 180
5.3.3 Методи тестового виявлення відмов 184
Контрольні питання до розділу 5 186
Рекомендована література 187

ПЕРЕДМОВА

Сучасні обчислювальні машини та системи є одним із найбільших досягнень наукової та інженерної думки , вплив якого на прогрес у всіх областях людської діяльності важко переоцінити. Особливо актуальними питаннями сьогодення є розвиток багатопроцесорних систем та розробка програмного забезпечення, яке б ефективно використовувало їх ресурси.

Традиційно до недавніх часів основною массою обчислювальних машин були однопроцесорні системи. Основними складовими еволюції таких систем, та зокрема їх процесоів, донедавна були в основному зростання тактової частоти процесора та деякі архітектурні вдосконалення. Проте при сучасному рівні мікротехнологій подальше збільшення тактової частоти є надзвичайно дорогим і неоправданим. Тому протягом останнього десятиління розвитку обчислювальної техніки розробники основну ставку роблять на вдосконалення архітектури процесора та "просування" паралелізму, а не на збільшення тактової частоти. Зазвичай, найчастіше з метою "нарощення" архітектури процесора розробники додають декілька додаткових обчислювальних ядер або процесорів. Такі рішення посилили і не без того великий розрив між розвитком апаратної частини ЕОМ та програмного забезпечення. Тому більшість сучасних програмних продуктів не здатно ефективно використовувати всю потужність таких систем.

Метою даного конспекта лекцій є ознайомлення читача з основними класами сучасних комп’ютерних систем, принципами їх організації та функціонування. У першій і другій частині наведено класифікацію комп’ютерних систем та розглянуто найбільш типові представники елементів класифікації, розглянуто топології багатопроцесорних систем. У третій частині розглянуто методи організації обчислень в складних багатопроцесоних системах. У четвертому – особливості організації систем введення-вивтедення та організації пам’яті. У п’ятому розділі розглянуто питання діагностики та експлуатації комп’ютених систем.

ПЕРЕЛІК ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

COMA – архітектура комп’ютера з кешуванням (Cache Only Memory Architecture);

DSM – система із розподіленою пам’яттю спільного використання(Distribute Shared Momory);

HDD – жорсткий диск (Hard Disk Drive);

JBOD – технологія об’єднання декількох дисків (Just A Bunch Of Disks);

NORMA – система без прямого доступу до віддаленої пам’яті (No Remote Memory Access);

NUMA – неоднорідний доступ до пам’яті (Non-Uniform Memory Access);

SISD – система з одним потоком команд і одним потоком даних (Single Instruction, Single Data);

SIMD – система з одним потоком команд і багатьма потоками даних (Single Instruction, Multiple Data);

MISD - система з багатьма потоками команд і одним потоком даних (Multiple Instruction, Single Data);

MIMD – система з багатьма потоками команд і багатьма потоками даних (Multiple Instruction, Multiple Data);

SMP – симетрична мультипроцесорна система (Symmetric Multi Processors);

MPP – система з масовою паралельною обробкою (Massively Parallel Processing);

RISC – архітектура комп’ютера із скороченим набором команд (Restricted (reduced) instruction set computer);

RAID – технологія підвищення надійності і ефективності жорстких дисків (Redundant Arrays of Inexpensive Disks)

АЗП – асоціативний запам’ятовуючий пристрій;

АЛП – арифметико-логічний пристрій;

АП – асоціативна пам’ять;

АТФ – клас ферментів;

БКЕ – базовий комутуючий елемент (β-елемент);

ГТІ – генератор тактових імпульсів;

ДНК – дезоксирибонуклеїнова кислота;

ЕОМ – електронна обчислювальна машина;

КС – комп’ютерна система;

КМП – контролер масиву процесорів;

КТ – контрольна точка;

ММЗ – мережа міжз’єднань;

МП – масив процесорів;

ОС – обчислювальна система;

ПЕ – процесорний елемент;

ПК – пристрій керування;

ПП – периферійний пристрій;

ПЕОМ – паралельна електронно-обчислювальна машина;

СВВ – система введення-виведення;

СУБД – система управління базами даних;

ЦП – центральний процесор;

ФБ – функціональний блок;

ЧПК – числа з плаваючою комою.

РОЗДІЛ 1. СТРУКТУРИ ПАРАЛЕЛЬНИХ І РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ (ПРКС)

Тема 1.1. Комп'ютерні системи та паралельна обробка інформації

1.1.1 Загальні принципи функціонування комп'ютерних систем

Початком історії зародження і розвитку обчислювальної техніки став 1833 рік, коли англійський математик Чарльз Беббідж вперше проникся ідеєю створення механічного «обчислювального помічника», використовуючи принцип програмного управління. Було потрібно більше 100 років, щоб ця ідея, доповнена американським математиком Джоном фон Нейманом в 1945 - 47 рр. і основана на електронних лампах, що з'явилися на той час, поклала початок ери ЕОМ. З моменту створення в 1947 р. першої програмно-керованої цифрової ЕОМ почався бурхливий прогрес обчислювальної техніки. Вдосконалення елементної бази і еволюція архітектурних рішень привело до істотного зменшення розмірів, вартості і енергоспоживання, а також до підвищення швидкодію і надійність ЕОМ. Великі успіхи досягнуті також в області периферійного устаткування, що істотно полегшило спілкування користувачів з ЕОМ і підвищило ємкість накопичувачів інформації.

Згідно принципам роботи ЕОМ, сформульованих фон Нейманом, центральний процесор складається з двох частин. Пристрій управління сприймає команди програм і організовує їх виконання. Арифметико-логічний пристрій виконує обчислення.

Дані зберігаються в різних запам'ятовуючих пристроях. Для довготривалого зберігання інформації використовуються постійні носії, які служать для введення даних і виведення результатів роботи. Для зберігання виконуваних в даний момент програм і проміжних даних використовується оперативна пам'ять, яка працює значно швидше від постійної пам'яті.

Взаємодія користувача з персональним комп’ютером – введення даних і надання результатів їх обробки реалізується широким спектром пристроїв введення-виведення.

Найважливіший компонент будь-якого персонального комп’ютера - це мікропроцесор (CPU, Central Processor Unit - ЦП), який управляє роботою комп’ютера і виконує велику частину обробки інформації. Мікропроцесор є надвеликою інтегральною схемою, ступінь інтеграції якої визначається розміром кристала і кількістю реалізованих в нім транзисторів. Іноді інтегральні мікросхеми називають чіпами (англ. chip). Базовими елементами мікропроцесора є транзисторні перемикачі, на основі яких будуються, наприклад, регістри, що є сукупністю пристроїв, що мають два стійкі стани і призначених для зберігання інформації і швидкого доступу до неї. Кількість і розрядність регістрів багато в чому визначають архітектуру мікропроцесора.

Виконувані мікропроцесором команди передбачають, як правило, арифметичні дії, логічні операції, передачу управління (умовну і безумовну) і пересилання даних (між регістрами, оперативною пам’яттю і портами введення/виводу). Із зовнішніми пристроями мікропроцесор може спілкуватися завдяки своїм шинам адреси, даних і управління.

Об’єм пам’яті, що фізично адресується мікропроцесором, однозначно визначається розрядністю зовнішньої адресної шини як 2N, де N - кількість адресних ліній.

Перші покоління комп’ютерів розроблялися на основі концепції жорсткої архітектури. Кожна модель містила унікальне обчислювальне ядро і орієнтувалася на обслуговування абсолютно певного набору каналів введення-виведення-виводу, що забезпечувалося спеціалізованим програмним забезпеченням. Довільна конфігурація моделі була неможлива, і будь-які функціональні модифікації могли бути здійснені тільки фірмою-розробником шляхом проведення комплексних доопрацювань архітектури комп’ютера.

У надрах останнього покоління комп’ютерів, що передувало епосі персональних комп’ютерів, народилася концепція *відкритої архітектури* (рис.1.1). Згідно цієї концепції, конкретна конфігурація моделі могла бути

визначена на стадії виробництва або навіть уточнена користувачем.

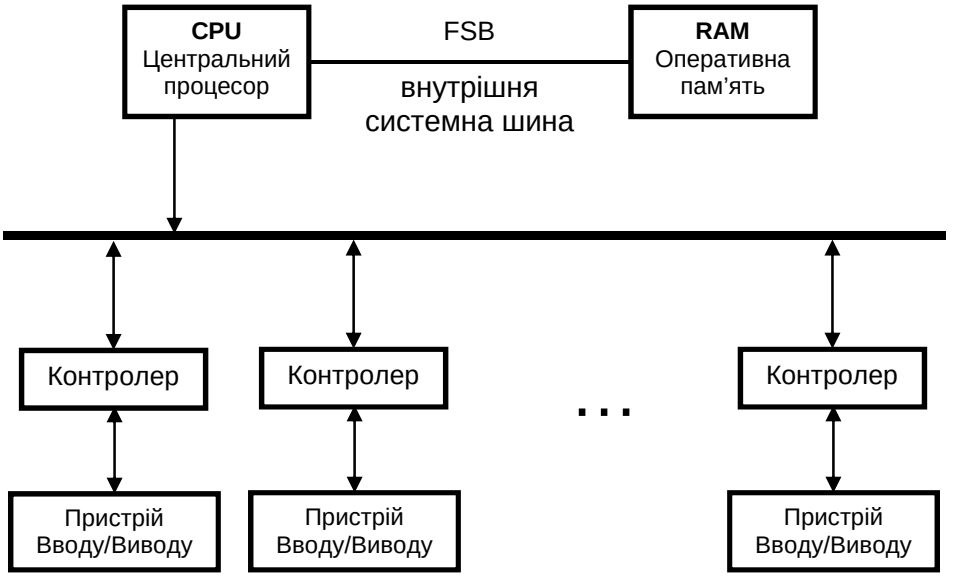


Рис. 1.1 Блок-схема комп'ютера з відкритою архітектурою

Відкрита архітектура передбачає строгу стандартизацію головних системних шин і свободу підключення до головної шини через відповідні пристрої управління (контроллери, карти) будь-якого (з деякими обмеженнями) набору пристроїв введення і виведення інформації, що розміщені у будь-якому порядку . Це забезпечує конфігурацію конкретного комп'ютера ніби із «кубиків» відповідно до його цільового призначення. З іншого боку, така концепція забезпечує розподіл праці при розробці окремих блоків і масовому виробництві комп'ютерів, їх сумісність, швидку і безболісну модернізацію, безперервне здешевлення на базі конкуренції.

З початку розвитку комп'ютерів і приблизно до 2000-2005 років провідні виробники мікропроцесорів досягали збільшення продуктивності останніх не на стільки за рахунок вдосконалення архітектурних рішень, як за рахунок збільшення тактової частоти процесорів. Але така тенденція на сьогоднішній день не є актуальною, оскільки подальше збільшення частоти при даному рівні технологій приводить до неоправдано високих затрат. Тому сучасна стратегія розвитку процесорів персональних ЕОМ пов'язана із збільшенням процесорів або обчислювальних ядер.

Виникнення терміну «ядро» пов'язане з модифікацією архітектури

процесора, що, у свою чергу, багато в чому визначається застосуванням технології виробництва мікропроцесорів з вищим рівнем інтеграції, що дає можливість реалізувати на кристалі нові технології обробки інформації.

Перший двохядерний процесор Power4 для серверів був анонсований фірмою IBM у 1999році, а у 2001році було розпочато їх продаж. У 2002 році AMD і Intel оголошують про перспективи створення своїх двохядерних процесорів;

Ідея багатоядерного процесора виглядає на перший погляд абсолютно тривіальною: просто упаковуємо два-три (або більш) процесори в один корпус - і комп'ютер отримує можливість виконувати декілька програмних потоків одночасно. Таке рішення є більш економічно оправданим, аніж застосування багатопроцесорних систем, проте часто є менш ефективним.

Якщо оцінювати розвиток комп'ютерних систем у глобальному масштабі, то розглянуті сучасні ЕОМ на основі багатопроцесорних та багатоядрних конфігурації є лише невеликою частиною всього ринку новітніх обчислювальних засобів. З метою систематизації всіх обчислювальних систем, доцільно їх класифікувати по певних ознаках.

1.1.2 Класифікація КС

Одним з найбільш поширених способів класифікації ЕОМ є *систематика Флінна* (Flynn), в рамках якої основна увага при аналізі архітектури обчислювальних систем приділяється способам взаємодії послідовностей (*потоків*) виконуваних команд і оброблюваних даних. В результаті такого підходу розрізняють наступні основні типи систем (рис. 1.2):

SISD (Single Instruction, Single Data) – системи, в яких існує одиночний потік команд і одиночний потік даних. У таких машинах є тільки один потік команд, всі команди обробляються послідовно один за одним і кожна команда ініціює одну операцію з одним потоком даних. До даного типу систем можна віднести звичайні послідовні ЕОМ;

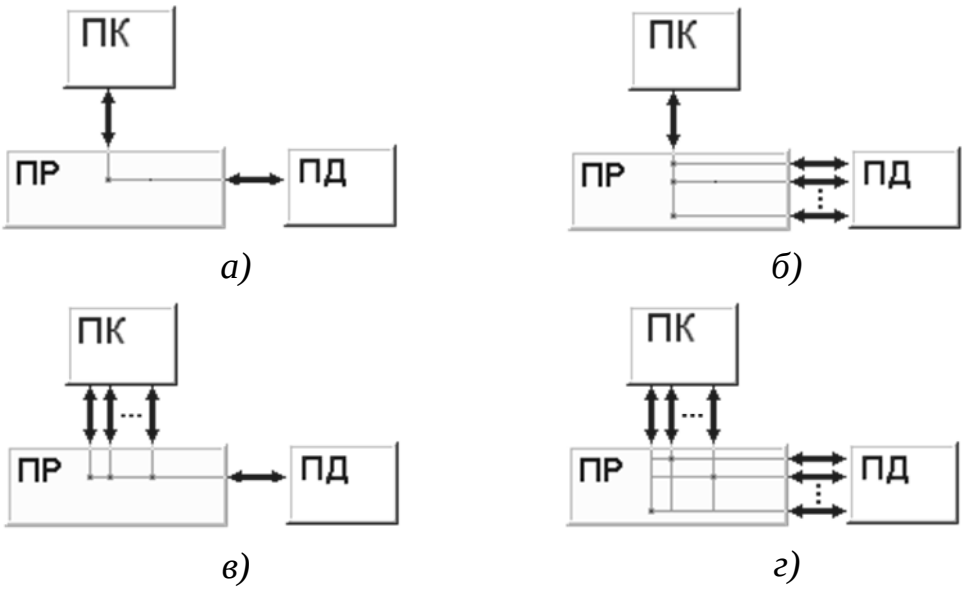


Рис. 1.2 Класифікація Флінна
а) - клас SISD; б)- клас SIMD; в)- клас MISD; г)- клас MIMD.
ПК – пристрій керування, ПР-Процесор, ПД- пам’ять даних

SIMD (Single Instruction, Multiple Data) – системи з одиночним потоком команд і множинним потоком даних. У архітектурі подібного роду зберігається один потік команд, що включає, на відміну від попереднього класу, векторні команди. Це дозволяє виконувати одну арифметичну операцію відразу над багатьма даними - елементами вектора. Подібною архітектурою володіють, наприклад, багатопроцесорні системи з єдиним пристроєм керування; даний підхід широко використовувався в попередні роки (системи ILLIAC IV, ICL DAP, Goodyear Aerospace MPP, Connection Machine), останнім часом його застосування обмежене, в основному, створенням спеціалізованих систем;

MISD (Multiple Instruction, Single Data) – системи, в яких існує множинний потік команд і одиночний потік даних; Визначення має на увазі наявність в архітектурі багатьох процесорів, що обробляють один і той же потік даних. Проте ні Флінн, ні інші фахівці в області архітектури комп’ютерів до цих пір не змогли представити переконливий приклад реально існуючої обчислювальної системи, побудованої на даному принципі. Вважатимемо, що даний клас порожній;

MIMD (Multiple Instruction, Multiple Data) – системи з множинним потоком команд і множинним потоком даних; до подібного класу систем відносяться більшість паралельних багатопроцесорних обчислювальних систем.

Слід зазначити, що хоча систематика Флінна широко використовується при конкретизації типів комп'ютерних систем, така класифікація приводить до того, що практично всі види паралельних систем (не дивлячись на їх істотну різнорідність) відносяться до однієї групи MIMD. Як результат, багатьма дослідниками робилися неодноразові спроби деталізації систематики Флінна.

Інший більш загальний спосіб класифікації комп'ютерних систем передбачає поділ архітектур по способу звернення процесорів до пам'яті.

Перший клас – це *комп'ютери із загальною пам'яттю.* Системи, побудовані за таким принципом, іноді називають мультипроцесорними системами або просто мультипроцесорами. В системі присутні декілька рівноправних процесорів, що мають однаковий доступ до єдиної пам'яті (рис. 1.3). Всі процесори "розділяють" між собою загальну пам'ять, звідси ще одна назва комп'ютерів цього класу – комп'ютери з пам'яттю, що розділяється. Всі процесори працюють з єдиним адресним простором: якщо один процесор записав значення 79 в слово за адресою 1024, то інший процесор, прочитавши слово, розташоване за адресою 1024, отримає значення 79.

Рис. 1.3 Архітектура комп'ютерних систем із спільною пам'яттю

Другий клас – це комп'ютери з розподіленою пам'яттю, які по аналогії з попереднім класом іноді називатимемо мультикомп'ютерними системами (рис. 1.4). По суті справи, кожний обчислювальний вузол є повноцінним комп'ютером з своїм процесором, пам'яттю, підсистемою введення-виведення, операційною системою. В такій ситуації, якщо один процесор запише значення 79 за адресою

1024, то це ніяк не вплине на те, що за тією ж адресою прочитає інший, оскільки кожний з них працює в своєму адресному просторі.

Рис. 1.4 Архітектура комп'ютерних систем із розподіленою пам'яттю

До комп'ютерів із загальною пам'яттю відносяться всі системи класу Symmetric Multi Processors (SMP). В SMP все, окрім декількох процесорів, в одному екземплярі: одна пам'ять, одна операційна система, одна підсистема введення-виведення. Слово "симетричний" в назві архітектури означає, що кожний процесор може робити все те, що і будь-хто інший. До речі, в даний час SMP часто розглядають як альтернативна назва для комп'ютерів із загальною пам'яттю, чому додатково сприяють два можливі варіанти розшифровки даної абревіатури: Symmetric Multi Processors і Shared Memory Processors.

Тема 1.2. КС класу SIMD

1.2.1 Векторні і векторно-конвеєрні КС

SIMD-системи були першими обчислювальними системами, що складаються з великого числа процесорів, і серед перших систем, де була досягнута продуктивність порядку GFLOPS. Згідно класифікації Флінна, до класу SIMD відносяться осбчислювальні системи (ОС), де безліч елементів даних піддається паралельній, але **однотипній** обробці. SIMD-системи багато в чому схожі на класичні фон-нейманівські системи: у них також є один пристрій управління, що забезпечує послідовне виконання команд програми. Відмінність стосується стадії виконання, коли загальна команда транслюється безлічі процесорів (у простому випадку — арифметико-логічний пристрій (АЛП)), кожен з яких обробляє свої дані.

Хоча продуктивність ОС загального призначення неухильно зростає, як і раніше залишаються задачі, що вимагають істотно більшої обчислювальної потужності. До них перш за все слід віднести задачі моделювання реальних процесів і об’єктів, для яких характерна обробка великих регулярних масивів чисел з плаваючою комою.

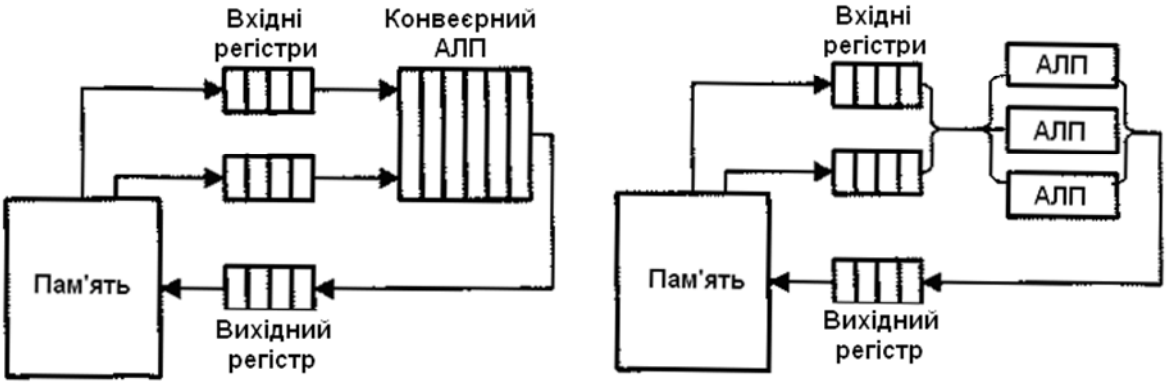
При великій розмірності масивів послідовна обробка елементів матриць займає дуже багато часу, що і приводить до неефективності універсальних ОС для даного класу завдань. Для обробки масивів потрібні обчислювальні засоби, що дозволяють за допомогою єдиної команди проводити дію відразу над всіма елементами масивів, — засоби *векторної обробки*.

Векторний процесор — це процесор, в якому операндами деяких команд можуть виступати впорядковані масиви даних — вектори. Векторний процесор може бути реалізований в двох варіантах. У першому він є додатковим блоком до універсальної обчислювальної машини (системи). У другому — векторний процесор — це основа самостійної ОС.

Розглянемо можливі підходи до архітектури засобів векторної обробки. Найбільш поширені з них зводяться до трьох груп:

- конвеєрний АЛП;
- масив АЛП;
- масив процесорних елементів.

Останній варіант — один з випадків багатопроцесорної системи, відомої як *матрична ОС*, що буде розглянуто в наступному пункті. Поняття векторного процесора має відношення до двох перших груп, причому, як правило, до першої. Ці дві категорії ілюструє рис. 1.4.



а) *б)*

Рис. 1.5 Варіанти векторних обчислень:
а — з конвеєрним АЛП; б — з декількома АЛП

У варіанті з конвеєрним АЛП (рис. 1.5, *а)* обробка елементів векторів проводиться конвеєрним АЛП для чисел з плаваючою комою (ПК). Операції з числами у формі з ПК достатньо складні, але піддаються розбиттю на окремі кроки. Так, додавання двох чисел може бути зведене до чотирьох етапів: порівнянню порядків, зсуву мантиси меншого з чисел, додаванню мантис і нормалізації результату (рис. 1.6, *а*). Кожен етап може бути реалізований за допомогою окремого ступеня конвеєрного АЛП (рис. 1.6, *б*). Черговий елемент вектора подається на вхід конвеєра, як тільки звільняється перший ступінь (рис. 1.6, в). Зрозуміло, що такий варіант цілком годиться для обробки векторів.

Одночасні операції над елементами векторів можна проводити і за

допомогою декількох паралельно використовуваних АЛП, кожне з яких відповідає за одну пару елементів (див. рис. 1.5, *б*). Такого роду обробка, коли кожне з АЛП є конвеєрним, показана на рис. 1.6, *г*.

Якщо паралельно використовуються конвеєрні АЛП, то можливий ще один рівень конвеєризації, що ілюструє рис. 1.6, *д*. Обчислювальні системи, де реалізована ця ідея, називають *векторно-конвеєрними*. Комерційні векторно-конвеєрні ОС, до складу яких для забезпечення універсальності включений також скалярний процесор, відомі як *суперЕОМ*.

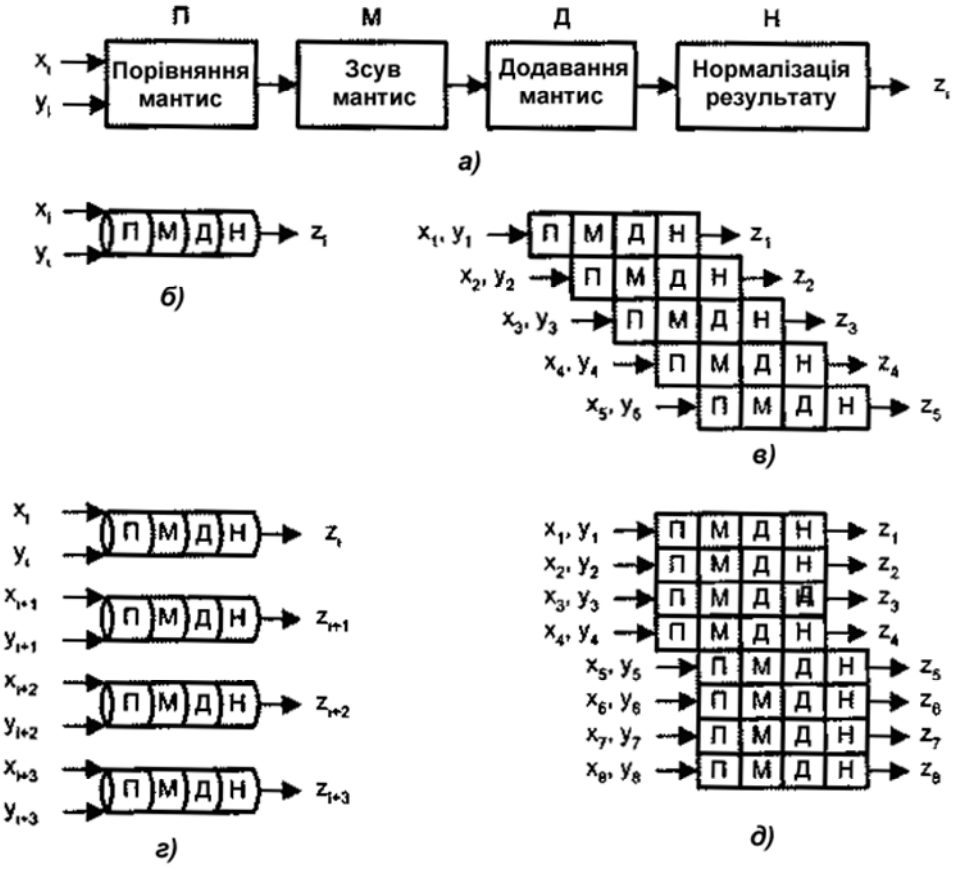


Рис. 1.6 Обробка векторів:
а — структура арифметичного конвеєра для чисел з плаваючою комою;
б — позначення конвеєра; *в* — обробка векторіву конвеєрному АЛП;
***г*— паралельна обробка векторів декількома конвеєрними АЛП;**
***д* — конвеєрна обробка векторів чотирма АЛП**

Узагальнена структура векторного процесора наведена на рис. 1.7. На схемі показані основні вузли процесора, без деталізації деяких зв’язків між ними.

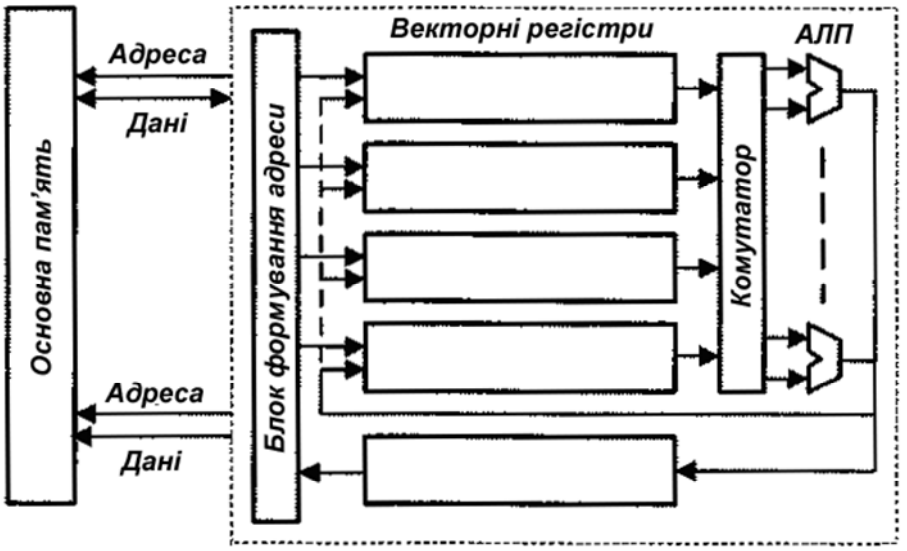


Рис. 1.7 Структура векторного процесора

Обробка всіх *n* компонентів векторів-операндів задається однією *векторною командою.* Загальноприйнято, що елементи векторів представляються числами у формі з плаваючою комою (ЧПК). АЛП векторного процесора може бути реалізоване у вигляді єдиного конвеєрного пристрою, здатного виконувати всі передбачені операції над числами з ПЗ. Проте, більш поширеніша інша структура, — в ній АЛП складається з окремих блоків додавання і множення, а іноді і блоку для обчислення зворотної величини, коли операція ділення *X/Y* реалізується у вигляді *X(1/Y)*. Кожен з таких блоків також конвеєризований.

Крім того, до складу *векторної обчислювальної системи* зазвичай включають і скалярний процесор, що дозволяє паралельно виконувати векторні і скалярні команди.

Для зберігання векторів-операндів замість безлічі скалярних регістрів використовують так звані *векторні регістри*, які є сукупністю скалярних регістрів, об’єднаних в чергу типу FIFO, здатну зберігати 50-100 чисел з плаваючою комою. Набір векторних регістрів (V_a, V_b, V_c...) є в будь-якому векторному процесорі. Система команд векторного процесора підтримує роботу з векторними регістрами і обов’язково включає команди:

- завантаження векторного регістра даними, які розміщені послідовно в

пам’яті, які задаються адресою першої комірки цієї послідовності;

- виконання операцій над всіма елементами векторів, що знаходяться у векторних регістрах;
- збереження вмісту векторного регістра в послідовності комірок пам’яті, вказаних адресою першої комірки цієї послідовності.

Принципова відмінність архітектури векторних процесорів полягає в тому, яким чином здійснюється доступ до операндів. При організації типу «*память-памятъ*» елементи векторів по черзі зчитуються з пам’яті і відразу ж прямують у функціональний блок. По мірі обробки елементи отримуваного вектора результату відразу ж заносяться в пам’ять. У архітектурі типу «*регістр-регістр*» операнди спочатку завантажуються у *векторні регістри*, кожен з яких може зберігати сегмент вектора, наприклад 64 елементи. Векторна операція реалізується шляхом зчитування операндів з векторних регістрів і занесення результату у векторні регістри.

Перевага процесора з режимом «пам’ять-пам’ять» полягає в можливості обробки довгих векторів, тоді як в процесорах типу «регістр-регістр» доводиться розбивати довгі вектори на сегменти фіксованої довжини. На жаль, за гнучкість режиму «пам’ять-пам’ять» доводиться розплачуватися відносно великими витратами, відомими як *час запуску*, що є часовим інтервалом між ініціалізацією команди і моментом, коли перший результат з’явиться на виході конвеєра. Великий час запуску в процесорах типу «пам’ять-пам’ять» обумовлений швидкістю доступу до пам’яті, яка набагато більше швидкості доступу до внутрішнього регістра. Проте коли конвеєр заповнений, результат формується в кожному циклі. Модель часу роботи векторного процесора має вигляд:

$$T=s + a \times N \qquad (1.1)$$

де *s* — час запуску, *a* — константа, залежна від команди (зазвичай 0.5, 1 або 2) і *N*, — довжина вектора.

Для підвищення швидкості обробки векторів всі функціональні блоки

векторних процесорів будуються за конвеєрною схемою, причому так, щоб кожен ступінь будь-якого з конвеєрів справлялася зі своєю операцією за один такт (число ступенів в різних функціональних блоках може бути різним). У деяких векторних ОС, наприклад Cray C90, цей підхід дещо вдосконалений - конвейери у всіх функціональних блоках продубльовані (рис. 1.8).

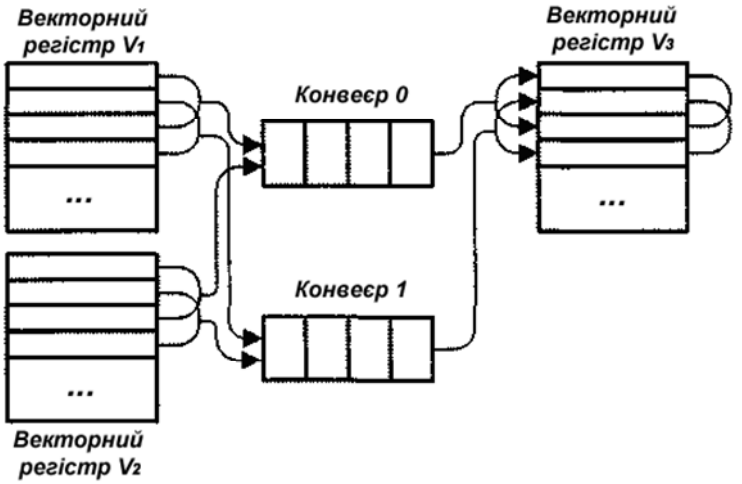


Рис. 1.8. Виконання векторних операцій при двох конвеєрах

На конвеєр 0 завжди подаються елементи векторів з парними номерами, а на конвеєр 1 – з непарними. У початковий момент на перший ступінь конвеєра 0 і векторних регістрів V1 і V2 поступають нульові елементи векторів. Одночасно перші елементи векторів з цих регістрів подаються на перший ступінь конвеєра 1. На наступному такті на конвеєр 0 подаються другі елементи з V1 і V2, а на конвеєр 2 — треті елементи і так далі Аналогічно відбувається розподіл результатів у вихідному векторному регістрі V3. У результаті функціональний блок при максимальному завантаженні в кожному такті видає не один результат, а два. Слід відмітити, що в скалярних операціях працює тільки конвеєр 0.

1.2.2 Матричні обчислювальні системи

Призначення *матричних обчислювальних систем* багато в чому схоже з призначенням векторних ОС — обробка великих масивів даних. В основі матричних систем лежить *матричний процесор* (array processor), що

складається з регулярного масиву процесорних елементів (ПЕ) Організація систем подібного типу на перший погляд достатньо проста. Вони мають загальний пристрій керування, який генерує потік команд, і велику кількість ПЕ, що працюють паралельно і кожен з них обробляє свій потік даних. Проте на практиці, щоб забезпечити достатню ефективність системи при вирішенні широкого кругу завдань, необхідно організувати зв’язки між процесорними елементами так, щоб найповніше завантажити процесори роботою. Саме характер зв’язків між ПЕ і визначає різні властивості системи. Раніше вже наголошувалося, що подібна схема може застосовуватись і для векторних обчислень.

Між матричними і векторними системами є істотна різниця. Матричний процесор інтегрує безліч ідентичних функціональних блоків (ФБ), що логічно об’єднаних в матрицю і працюють в SIMD-стилі. Не так суттєво, як конструктивно реалізована матриця процесорних елементів — на єдиному кристалі, чи на декількох. Важливий сам принцип — ФБ логічно скомпоновані в матрицю і працюють синхронно, тобто присутній тільки один потік команд для всіх. Векторний процесор має вбудовані команди для обробки векторів даних, що дозволяє ефективно завантажити конвеєр з функціональних блоків. У свою чергу, векторні процесори простіше використовувати, тому що команди для обробки векторів — це зручніша для людини модель програмування, ніж SIMD.

Структуру матричної обчислювальної системи можна наведено на рис.1.9.

Власне паралельна обробка множинних елементів даних здійснюється *масивом процесорів* (МП). Єдиний потік команд, що керує обробкою даних в масиві процесорів, генерується *контролером масиву.*

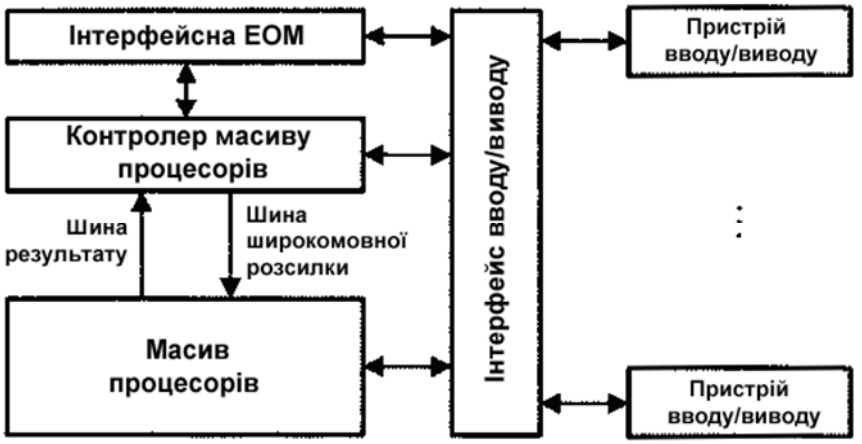


Рис. 1.9. Узагальнена модель матричної SIMD-системи *процесорів.*

Контролер масиву процесорів (КМП) виконує послідовний програмний код, реалізує операції умовного і безумовного переходів, транслює в МП команди, дані і сигнали управління. Команди обробляються процесорами в режимі жорсткої синхронізації. Сигнали управління використовуються для синхронізації команд і пересилок, а також для керування процесом обчислень, зокрема визначають, які процесори масиву повинні виконувати операцію, а які — ні. Команди, дані і сигнали управління передаються з КМП в масив процесорів по *шині широкомовної розсилки.* Оскільки виконання операцій умовного переходу залежить від результатів обчислень, результати обробки даних в масиві процесорів транслюються в КМП, проходячи по *шині результату.*

Для забезпечення користувача зручним інтерфейсом при створенні і відладці програм до складу подібних ОС зазвичай включають *інтерфейсні ЕОМ* (front-end computer). В ролі такої ЕОМ виступає універсальна обчислювальна машина, на яку додатково покладається завдання завантаження програм і даних в КМП. Крім того, завантаження програм і даних в КМП може проводитися і безпосередньо з *пристроїв вводу/виводу*, наприклад з магнітних дисків. Після завантаження КМП приступає до виконання програми, транслюючи в МП по широкомовній шині відповідні SIMD-команди.

Розглядаючи масив процесорів, слід враховувати, що для зберігання

множинних наборів даних в ньому, крім безлічі процесорів, повинно бути присутнім і безліч модулів пам’яті. Крім того, в масиві повинна бути реалізована мережа взаємозв’язків, як між процесорами, так і між процесорами і модулями пам’яті. Таким чином, під терміном *масив процесорів* розуміють блок, що складається з процесорів, модулів пам’яті і мережі з’єднань.

Додаткову гнучкість при роботі з даною системою забезпечує механізм *маскування,* що дозволяє залучати до операцій лише певну підмножину з процесорів, які входять в масив. Маскування реалізується як на стадії компіляції, так і на етапі виконання, при цьому процесори, виключені шляхом установки в нуль відповідних бітів маски, під час виконання команди простоюють.

У матричних SIMD-системах поширення набули два основні типи архітектурної організації масиву процесорних елементів (рис. 1.10).

У першому варіанті, відомому як архітектура типу «*процесорний елемент-процесорний елемент*» («ПЕ-ПЕ»), N процесорних елементів (ПЕ) зв’язані між собою мережею з’єднань (рис. 1.10, *а).* Кожен ПЕ — це процесор з локальною пам’яттю. Процесорні елементи виконують команди, що отримуються з КМП по шині широкомовної розсилки, і обробляють дані, які можуть зберігатися в їх локальній пам’яті, або поступати із з КМП. Обмін даними між процесорними елементами проводиться по *мережі з’єднань,* тоді як *шина вводу/виводу* служить для обміну інформацією між ПЕ і пристроями вводу/виводу. Для трансляції результатів з окремих ПЕ в контроллер масиву процесорів служить *шина результату.* Завдяки використанню локальної пам’яті апаратні засоби ОС даного типу можуть бути побудовані досить ефективно. У багатьох алгоритмах операції з пересилки даних здебільшого локальні, тобто відбуваються між найближчими сусідами. З цієї причини архітектура, де кожен ПЕ пов’язаний тільки з сусідніми, дуже популярна. Як приклади обчислювальних систем з даною архітектурою можна згадати MasPar MP-1, Connection Machine CM-2, GF11, DAP, MPP, STARAN, ILLIAC.

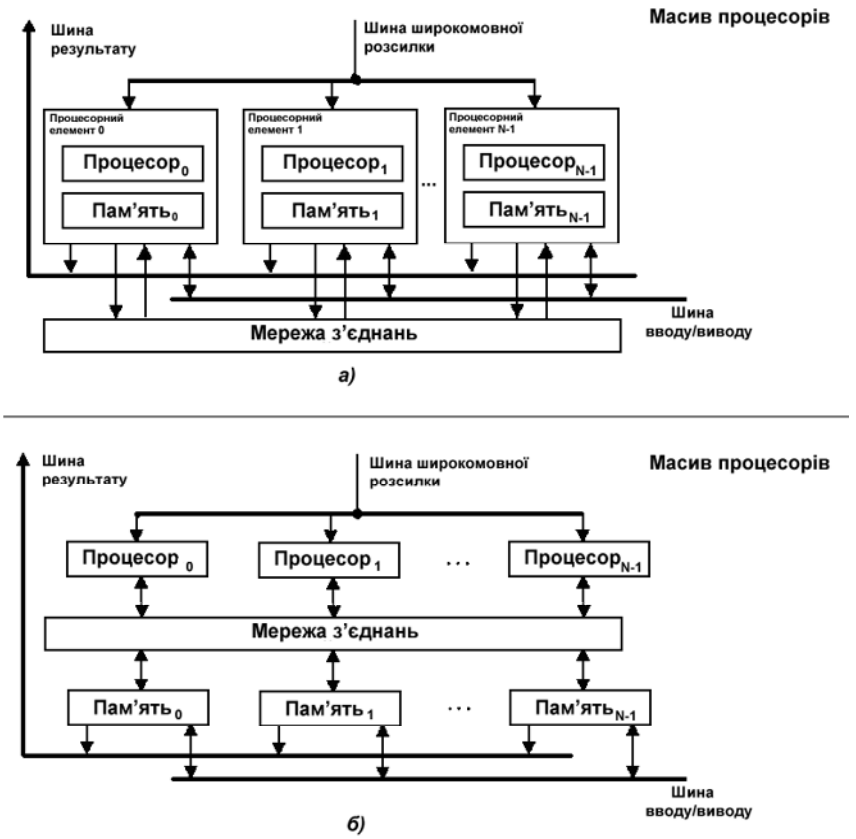


Рис. 1.10. Моделі масивів процесорів, *а* — «процесорний елемент-процесорний елемент»; *б* — «процесор-пам’ять»

Другий вид архітектури — «*процесор-пам’ять*» — показаний на рис.1.10,*б*. У такій конфігурації двонаправлена мережа з’єднань пов’язує *N* процесорів з *M* модулями пам’яті. Процесори управляються КМП через широкомовну шину. Обмін даними між процесорами здійснюється як через мережу, так і через модулі пам’яті. Пересилка даних між модулями пам’яті і пристроями вводу/виводу забезпечується шиною вводу/виводу. Для передачі даних із конкретного модуля пам’яті в КМП служить шина результату. Прикладами ОС з розглянутою архітектурою можуть служити Burroughs Scientific Processor (BSP), Texas Reconfigurable Array Computer TRAC.

Ефективність мереж взаємозв’язків процесорних елементів багато в чому визначає можливу продуктивність всієї матричної системи. Застосування знаходять найрізноманітніші топології мереж.

Оскільки процесорні елементи в матричних системах функціонують синхронно, обмінюватися інформацією вони також повинні по узгодженій

схемі, причому необхідно забезпечити можливість синхронної передачі від декількох ПЕ-джерел до одного ПЕ-приймача. Коли для передачі інформації в мережевому інтерфейсі задіюється тільки один регістр пересилки даних, це може привести до втрати даних, тому у ряді ОС для запобігання подібній ситуації передбачені спеціальні механізми. Так, в системі СМ-2 використовується устаткування, що об’єднує повідомлення, які поступили до одного ПЕ.

Об’єднання реалізується за рахунок операцій арифметичного і логічного додавання, накладення записів, знаходження меншого і більшого з двох значень. У деяких SIMD-системах, наприклад МР-1, є можливість записати повідомлення, що одночасно прийшли в різні елементи локальної пам’яті.

1.2.3 КС нетрадиційної архітектури

Асоціативні КС

Асоціативний спосіб обробки даних дозволяє подолати багато обмежень, властивих адресному доступу до пам’яті, за рахунок завдання деякого *критерію відбору* і проведення необхідних перетворень тільки над тими даними, які задовольняють цьому критерію. *Критерієм відбору* може бути співпадіння з будь-яким елементом даних, достатнім для виділення шуканих даних з тих, що є. Пошук даних може відбуватися по фрагменту, що має більшу або меншу кореляцію із заданим елементом даних.

Досліджені і по різному застосовуються декілька підходів, що розрізняються повнотою реалізації моделі *асоціативної обробки*. Якщо реалізується тільки асоціативна вибірка даних з подальшим почерговим використанням знайдених даних, то говорять про асоціативну *пам’ять* або пам’ять, що адресується по вмісту. При достатньо повній реалізації всіх властивостей асоціативної *обробки* використовується термін "асоціативний процесор".

Асоціативні системи відносяться до класу: один потік команд – безліч потоків даних (SIMD = Single Instruction Multiple Data). Ці системи включають велике число операційних пристроїв, здатних одночасно по командах керуючого пристрою вести обробку декількох потоків даних. У асоціативних обчислювальних системах інформація на обробку поступає від асоціативних запам’ятовуючих пристроїв (АЗП), які характеризуються тим, що інформація в них вибирається не за певною адресою, а по її змісту.

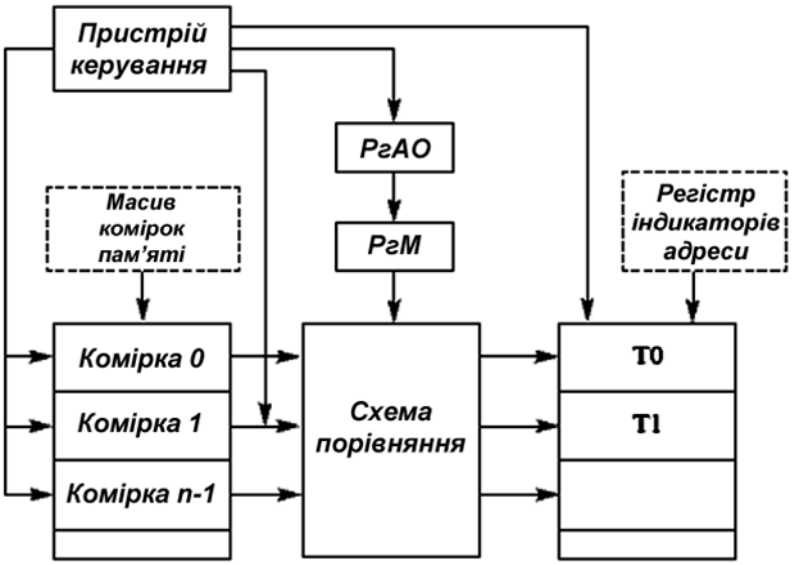


Рис. 1.11 Схема асоціативної системи

Головна відмінність асоціативної ОС (рис. 1.11) від звичайної системи послідовної обробки інформації полягає у використанні асоціативної пам’яті або подібного пристрою, а не пам’яті з адресованими комірками пам’яті.

Асоціативна пам’ять (АП) допускає звернення за даними на основі їх *ознаки* або *ключового слова*: імені, набору характеристик, завдання діапазону і так далі.

Поширений вид АП — таблиця з двома стовпцями: "*поле запиту — поле результату*. Рядок таблиці займає регістр пам’яті. По ключовому слову обробляються поля запиту: проводиться пошук на основі порівнянь і видається результат з одного (або більше) полів результату. За допомогою маскування можна виділити тільки ті поля в ключовому слові, які потрібно використовувати при пошуку для порівняння.

Типовими операціями порівняння, що виконуються асоціативною пам’яттю, є: "рівно — не рівно", "найближче менше ніж — найближче більш ніж", "не більш ніж — не менше ніж", "максимальна величина — мінімальна величина", "в межах — за межами", "наступна величина більша — наступна величина менше" і т.д. Тобто все це є операції відношення і визначення належності.

Оскільки асоціативні ОС характеризуються тільки активним використанням АП в обчисленнях, то в цілому ці ОС володіють звичайними властивостями, можуть проводити складні перетворення даних і належати типу SIMD (STARAN, PEPE) або MIMD. Для паралельного звернення (для прискорення пошуку) АП розбита на модулі (32 модулі — в системі STARAN).

Систолічні КС

Під структурою систолічної системи можна розуміти масив (мережа) обчислювальних «клітин» (ПЕ), зв’язаних каналами обміну. Число сусідніх клітин обмежене. Кожна клітина працює за своєю програмою. Обмін даними і сигналами керування здійснюється тільки з сусідніми клітинами (). Обмін з оперативною пам’яттю відбувається тільки на межі масиву клітин. Термін «систола» (systolic) указує на синхронність, безперервність і хвилеподібність просування оброблюваних даних від однієї межі масиву до іншої (систола – це серцевий м’яз, який працює також синхронно, ритмічно і без зупинок в ході всього життя людини, система як систоли «прокачує» через себе оброблювані дані). Особливістю роботи структур систол є відсутність накопичення інформації в локальних блоках пам’яті і повернення потоків даних назад для циклічної обробки. Впорядкованість етапів обробки інформації указує на спорідненість з конвеєрною архітектурою типу MISD, тому багато дослідників відносять системи систолічні ситеми саме до цьому класу. В той же час наявність різних

потоків команд і даних в системі і різних програм у різних обчислювальних клітин указує на спорідненість з MIMD-архитектурами. В той же час на відміну від MIMD характер обміну носить заданий, однонаправлений характер, відсутність циклів, повернень, довгих пересилок, а також відносна простота завдань, що вирішуються кожною обчислювальною клітиною, дозволяє говорити про своєрідність структур систол. Структура систолічної на ведена на рис. 1.12.

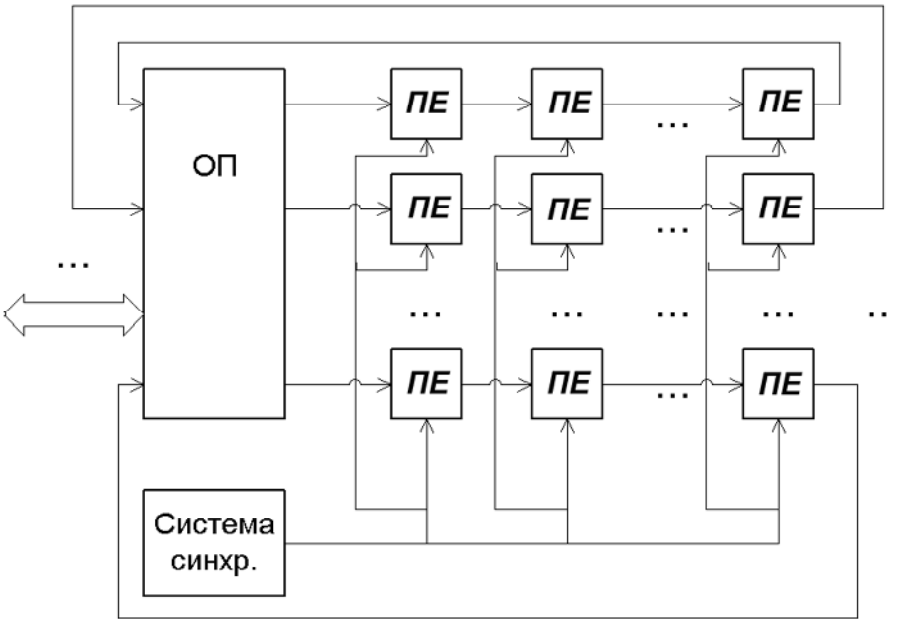


Рис. 1.12 Загальна структура систолічної системи

Областю застосування таких структур перш за все є багатопроцесорні спеціалізовані структури для цифрової обробки сигналів, зображень, для вирішення матричних задач.

Для ефективної реалізації обчислень в структурі систоли необхідні так звані алгоритми систол, розраховані на апаратну реалізацію систоли. Вони повинні задовольняти певним вимогам, серед яких :

- Регулярність, однонаправленість графа обчислень (потокового графа) алгоритму;
- Ациклічність алгоритму;
- Можливість розбиття алгоритму на етапи однакової складності і тривалості виконання для побудови конвеєра;

- Можливість розпаралелювання обчислень;
- Відсутність необхідності у великих об’ємах пам’яті для збереження проміжних результатів і накопичення інформації;
- Локальність пересилок інформації, відсутність необхідності в довгих пересилках;
- Мінімальна кількість розгалуджень в алгоритмі;
- Мінімальна кількість вхідних і вихідних точок алгоритму;
- Мінімальна кількість різних типів обчислень і операцій, використовуваних в алгоритмі;
- Можливість розбиття алгоритму на підалгоритми меншої розмірності, і з іншого боку – нарощування алгоритму для вирішення завдань більшої розмірності;
- Гарантована збіжність обчислень за задане число кроків (ітерацій).

Обчислювальні системи з командними словами надвеликої довжини (VLIW)

Архітектура з командними словами надвеликої довжини або *з наддовгими командами (*VLIW, Very Long Instruction Word) відома з початку 80-х з ряду університетських проектів, але тільки зараз, з розвитком технології виробництва мікросхем вона знайшла своє гідне втілення. VLIW — це набір команд, організованих на зразок горизонтальної мікрокоманди в мікропрограмному пристрої керування.

Ідея VLIW базується на тому, що завдання ефективного планування паралельного виконання декількох команд покладається на «розумний» компілятор. Такий компілятор спочатку досліджує початкову програму з метою виявлення всіх команд, які можуть бути виконані одночасно, причому так, щоб це не приводило до виникнення конфліктів. В процесі аналізу компілятор може навіть частково імітувати виконання даної програми. На наступному етапі компілятор намагається об’єднати такі команди в пакети,

кожен з яких розглядається як одна наддовга команда. Об’єднання декількох простих команд в одну наддовгу проводиться по наступних правилах:

- кількість простих команд, що об’єднуються в одну команду надвеликої довжини, рівна числу наявних в процесорі функціональних блоків (ФБ);
- у наддовгу команду входять тільки такі прості команди, які виконуються різними ФБ, тобто забезпечується одночасне виконання всіх складових наддовгої команди.

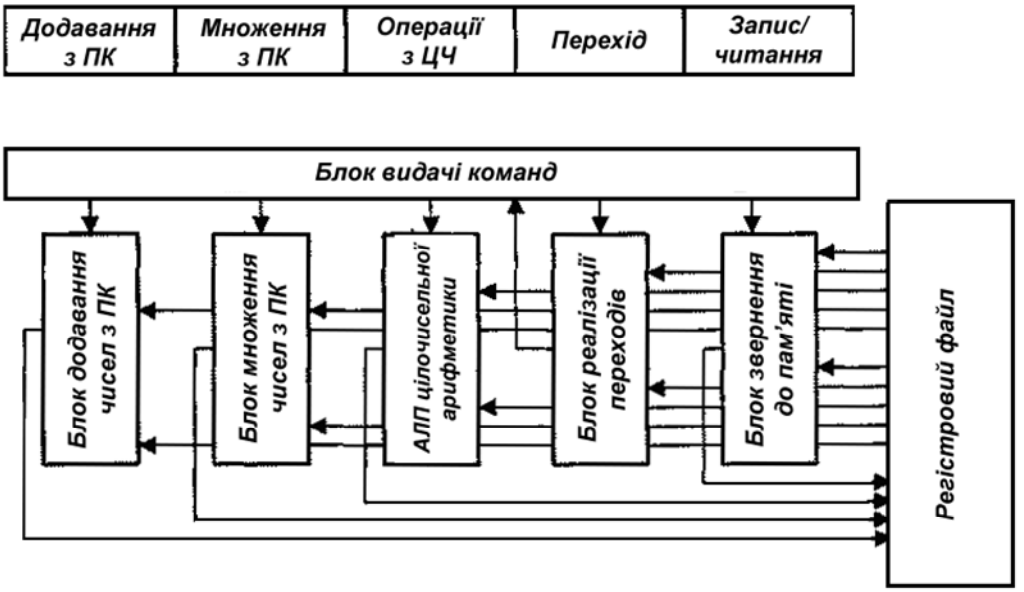


Рис. 1.13 Формат наддовгої команди і взаємозв’язок полий команди з складовими блоку виконання

Довжина наддовгої команди зазвичай складає від 256 до 1024 біт. Така *метакоманда* містить декілька полів (по кількості створюючих її простих команд), кожне з яких описує операцію для конкретного функціонального блоку. На рис. 1.13 показаний можливий формат наддовгої команди і взаємозв’язок між її полями і ФБ, що реалізовують окремі операції.

Як видно з малюнка, кожне поле наддовгої команди відображається на свій функціональний блок, що дозволяє отримати максимальну віддачу від апаратури блоку виконання команд.

VLIW-архітектуру можна розглядати як статичну суперскалярну

архітектуру. Мається на увазі, що розпаралелювання коду проводиться на етапі компіляції, а не динамічно під час виконання. Те, що у виконуваній наддовгій команді виключена можливість конфліктів, дозволяє гранично спростити апаратуру VLIW-процесора і, як наслідок, добитися вищої швидкодії.

В якості простих команд, які утворюють наддовгу, зазвичай використовуються команди RISC-типу, тому архітектуру VLIW іноді називають пост RISC-архітектурою. Максимальне число полів в наддовгій команді рівне числу обчислювальних пристроїв і зазвичай коливається в діапазоні від 3 до 20. Всі обчислювальні пристрої мають доступ до даних, що зберігаються в єдиному багатопортовому регістровому файлі. Відсутність складних апаратних механізмів, характерних для суперскалярних процесорів (прогноз переходів, позачергового виконання і т. д.), дає значний виграш в швидкодії і можливість ефективніше використовувати площу кристала. Переважна більшість цифрових сигнальних процесорів і мультимедійних процесорів з продуктивністю більше 1 млрд. операцій/с базується на VLIW-архитектуре. Більшість сучасних відеокарт використовує архітектуру VLIW для збільшення продуктивності роботи графічного процесора. Серйозна проблема VLIW — ускладнення регістрового файлу і зв’язків цього файлу з обчислювальними пристроями.

Клітинні та ДНК комп’ютери

В даний час у пошуках реальної альтернативи напівпровідниковим технологіям створення нових обчислювальних систем учені звертають все більшу увагу на біотехнології, або ***біокомпютінг***, який є гібридом інформаційних, молекулярних технологій, також біохімії. Біокомпьютінг дозволяє вирішувати складні обчислювальні завдання, користуючись методами, прийнятими в біохімії і молекулярній біології, організовуючи обчислення за допомогою живих тканин, кліток, вірусів і біомолекул.

Найбільшого поширення набув підхід, де як основний елемент (процесор) використовуються молекули дезоксирибонуклеїнової кислоти. Центральне місце в цьому підході займає так званий ДНК - процесор. Окрім ДНК як біо-процесори можуть бути використані також білкові молекули і біологічні мембрани.

Так само, як і будь-який інший процесор, ДНК процесор характеризується структурою і набором команд. У нашому випадку структура процесора - це структура молекули ДНК. А набір команд - це перелік біохімічних операцій з молекулами. Принцип пристрою комп'ютерної ДНК-пам'яті заснований на послідовному з'єднанні чотирьох нуклеотидов (основного блока ДНК-ланцюга). Три нуклеотиди, з'єднуючись в будь-якій послідовності, утворюють елементарний елемент пам'яті - кодон, які потім формують ланцюг ДНК. Основна трудність в розробці ДНК-комп'ютера пов'язана з проведенням вибірки однокодонних реакцій (взаємодій) усередині ланцюга ДНК. Проте прогрес є вже і в цьому напрямі. Вже є експериментальне устаткування, що дозволяє працювати з одним з 1020 кодонів або молекул ДНК. Іншою проблемою є самогрупування ДНК, що приводить до втрати інформації. Її долають введенням в клітину спеціальних інгібіторів - речовин, що запобігають хімічній реакції самогрупування.

Ідея використання ДНК для зберігання і обробки інформації виникла в 1994 році, коли американський дослідник Leonard Adleman з University of Southern California запропонував використовувати цю молекулу для вирішення простого математичного завдання. Поступово розвиваючи цей напрям науки, учені наближають епоху, коли живі "обчислювальні машини" зможуть уміщатися в одній клітці живого організму. Подібний "біологічний нанокомп'ютер" буде настільки малий, що трильйон таких обчислювальних пристроїв зможуть працювати одночасно в краплі води. Теоретичні розрахунки дають підстави припускати, що так званий ДНК-комп'ютер здатен перевершити кремнієві чіпи при вирішенні завдань, що вимагають

одночасного виконання безлічі схожих операцій. Але ще принадніші перспективи біологічні нанокомп’ютери обіцяють в області медицини і фармакології.

Звичайно, пристрої подібного роду чекає ще дуже довгий шлях від теоретичних розробок і експериментів до застосування на практиці. Проте дослідження групи ізраїльських учених, що опублікували в 2001 році статтю в журналі "Nature", продемонстрували, що нескладні програмовані обчислювальні пристрої, створені на основі ДНК здатні працювати усередині живої клітини, стали об’єктивною реальністю. Зрештою загальна обчислювальна потужність біологічних комп’ютерів в ізраїльській моделі складала мільярд операцій в секунду при точності обчислень більше 99,8%. Витрати ж енергії на цю роботу обчислювалися однією мільярдною часткою вата.

У 2003 році в онлайновому виданні "Proceedings of the National Academy of Sciences" співробітники лабораторії біомолекулярних комп’ютерів Weizmann Institute of Science (Ізраїль) на чолі з професором Шапіро оголосили про створення нової моделі біомолекулярної машини, яка взагалі не вимагає зовнішнього джерела енергії і працює в 50 разів швидше, ніж її попередники. Раніші системи залежали від молекул АТФ, які є головним джерелом енергії клітинних реакцій. Але в останній моделі молекула ДНК забезпечує і обробку даних, і достатню кількість енергії для виконання операцій. Новий ДНК-комп’ютер, здатний проводити 330 трильйонів обчислювальних операцій в секунду, був внесений до Книги рекордів Гіннесу як "найменший біологічний обчислювальний пристрій, коли-небудь створений людиною".

Також за останні роки почали активно проводитися дослідження по створенню клітинних комп’ютерів, які є колоніями різних "розумних" самоорганізуючихся мікроорганізмів, в геном яких вдалося включити якусь логічну схему, яка могла б активізуватися у присутності певної речовини. Для цієї мети ідеально підійшли б бактерії, стакан з якими і був би

комп’ютером. Такі комп’ютери дуже дешеві у виробництві. Їм не потрібна така стерильна атмосфера, як при виробництві напівпровідників.

Головною властивістю комп’ютера такого роду є те, що кожна їх клітина є мініатюрною хімічною лабораторією. Якщо біоорганізм запрограмований, то він просто проводить потрібні речовини. Досить виростити одну клітку, що володіє заданими якостями, і можна легко і швидко виростити тисячі кліток з такою ж програмою.

Основна проблема, з якою стикаються творці клітинних біокомп’ютерів, - організація всіх кліток в єдину працюючу систему. На сьогоднішній день практичні досягнення в області клітинних комп’ютерів нагадують досягнення 20-х років в області ламповых і напівпровідникових комп’ютерів. Зараз в Лабораторії штучного інтелекту Массачусетського технологічного університету створена клітина, здатна зберігати на генетичному рівні 1 біт інформації. Також розробляються технології, що дозволяють одиничній бактерії відшукувати своїх сусідів, утворювати з ними впорядковану структуру і здійснювати масив паралельних операцій.

У 2001 р. американські учені створили трансгенні мікроорганізми (тобто мікроорганізми з штучно зміненими генами), клітки яких можуть виконувати логічні операції І та АБО. Фахівці лабораторії Оук-рідж, штат Теннесі, використовували здатність генів синтезувати той або інший білок під впливом певної групи хімічних подразників. Учені змінили генетичний код бактерій Pseudomonas putida таким чином, що їх клітки знайшли здатність виконувати прості логічні операції. Наприклад, при виконанні операції І в клітину подаються дві речовини (по суті - вхідні операнди), під впливом яких ген виробляє певний білок. Тепер учені намагаються створити на базі цих кліток складніші логічні елементи, а також подумують про можливість створення клітини, що виконує паралельні декілька логічних операцій.

Потенціал біокомп’ютерів дуже великий. До переваг, що вигідно відрізняють їх від комп’ютерів, заснованих на кремнієвих технологіях, відносяться:

- простіша технологія виготовлення, що не вимагає для своєї реалізації таких жорстких умов, як при виробництві напівпровідників;
- використання не бінарного, а тернарного коди (інформація кодується трійками нуклеотидов), що дозволить при меншій кількості кроків перебрати більше число варіантів при аналізі складних систем;
- потенційно виключно висока продуктивність, яка може складати до 1014 операцій в секунду за рахунок одночасного вступу до реакції трильйонів молекул ДНК;
- можливість зберігати дані з щільністю, що перевищує в трильйони разів показники оптичних дисків;
- виключно низьке енергоспоживання;

Проте, разом з очевидними перевагами, біокомп’ютери мають і істотні недоліки, такі як:

- складність із зчитуванням результатів - сучасні способи визначення кодуючої послідовності не досконалі, складні, трудомісткі і дорогі;
- низька точність обчислень, пов’язана з виникненням мутацій, прилипанням молекул до стінок посудин і так далі;
- неможливість тривалого зберігання результатів обчислень у зв’язку з розпадом ДНК протягом часу.

Хоча до практичного використання біокомп’ютерів ще дуже далеко, і вони навряд чи будуть розраховані на широкі маси користувачів, передбачається, що, вони знайдуть гідне застосування в медицині і фармакології, а також з їх допомогою стане можливим об’єднання інформаційних і біотехнології.

Тема 1.3. Теорія обчислювальних систем

1.3.1 Основні поняття ТОС

Теорія обчислювальних систем (ТОС)— інженерна дисципліна, що поєднує методи вирішення задач проектування й експлуатації ЕОМ, обчислювальних комплексів, систем і мереж.

Предмет теорії. Предметом дослідження в теорії обчислювальних систем є обчислювальні системи в аспектах їхньої продуктивності, надійності і вартості. У системі виділяються наступні складові:

1) технічні засоби, обумовлені конфігурацією системи — складом пристроїв і структурою зв’язків між ними;

2) режим обробки, що визначає порядок функціонування системи;

3) робоче навантаження, що характеризує клас оброблюваних задач і порядок їхнього надходження в систему.

Коли ЕОМ, обчислювальний комплекс, система чи мережа досліджується в цілому, як органічна єдність складових у взаємодії з навколишнім середовищем, і при цьому виявляються загальносистемні властивості і характеристики, говорять, що дослідження проводиться на системному рівні. Представлення досліджуваних об’єктів (ЕОМ, комплекси, системи і мережі) на системному рівні — найбільш характерна риса теорії обчислювальних систем.

Предметом дослідження може бути функціонування процесора, зовнішнього запам’ятовуючого пристрою і каналу введення — виведення, обмін даними між рівнями пам’яті, планування, обробка, системне введення вивід та інше. При цьому властивості елементів і підсистем вивчаються стосовно до цілям дослідження всієї системи, наприклад до оцінки продуктивності, і розглядаються як частини системи, що функціонують у взаємодії з іншими частинами.

1.3.2 Задачі ідентифікації та синтезу комп’ютерних систем

Задачі аналізу. Аналіз обчислювальних систем — визначення властивостей, властивій чи системі класу систем. Типова задача аналізу — оцінка продуктивності і надійності систем із заданою конфігурацією, режимом функціонування і робочим навантаженням. Інші приклади задач – визначення (оцінка) імовірності конфлікту при доступі до загальної шини, розподілу тривалості зайнятості процесора, завантаження каналу введення-виведення.

У загальному випадку задача аналізу формулюється в такий спосіб. Виходячи з мети дослідження призначається набір характеристик $Y=\{y_1, \ldots, y_{m1}\}$ досліджуваного об’єкта (обчислювальна система, її елемент, підсистема, деякий процес і ін.) і точність $\Delta = \{\delta_1, .., \delta_m\}$, з яким вони повинні бути визначені. Потрібно знайти спосіб оцінки характеристик Y об’єкта з заданою точністю Δ і на основі цього способу визначити характеристики.

При аналізі систем у процесі експлуатації оцінка характеристик Y здійснюється, як правило, вимірюванням параметрів функціонування з обробкою вимірювальних даних. У цьому випадку використовується методика, що встановлює склад вимірюваних параметрів, періодичність і тривалість вимірів, а також вимірювальні засоби і засоби обробки даних З метою скорочення витрат на аналіз прагнуть вимірювати по можливості менше число найбільш просто вимірюваних параметрів $X=\{x_1,... ,x_v\}$, а необхідний набір характеристик визначати непрямим методом — обчисленням з використанням залежностей $y_\tau= \varphi_\tau(X), \tau=1,...,m$. Ці залежності, або мають статистичну природу, або створюються на основі фундаментальних закономірностей теорії обчислювальних систем.

При аналізі проектованих систем для оцінки характеристик Y необхідно володіти моделлю F, що встановлює залежність $Y=F(X)$ характеристик від параметрів системи X, що визначають її конфігурацію, режим функціонування, робоче навантаження. У цьому випадку розв’язок задачі

зводиться до проведення експериментів на основі моделі, що дозволяють дати відповіді на цікавлячі питання. Точність оцінки характеристик проектованої системи залежить від адекватності моделі і похибки виміру параметрів *X*.

Задачі ідентифікації. При експлуатації обчислювальних систем виникає необхідність у підвищенні їхньої ефективності шляхом підбору конфігурації і режиму функціонування, що відповідають класу розв’язуваних задач і вимогам до якості обслуговування користувачів. У зв’язку з зростанням навантаження на систему і переходом на нову технологію обробки даних може знадобитися зміна конфігурації системи, використовування більш сучасних операційних систем і реалізованих ними режимів обробки. У цих випадках варто оцінити можливий ефект, для чого необхідні моделі продуктивності і надійності системи. Побудова моделі системи на основі апріорних даних про її організацію до даних вимірів називається ідентифікацією системи.

Порядок ідентифікації обчислювальної системи ілюструється рис.1.14. Відповідно до природи досліджуваних явищ для їхнього представлення пропонується функціональна модель, що описує явища з точністю до значень параметрів функцій. Процес створення такої моделі називається функціональною ідентифікацією системи. Як функціональні моделі можуть використовуватися різні математичні системи — диференціальні й алгебраїчні рівняння, мережі масового обслуговування й ін., що адекватно представляють досліджувані аспекти.

Після того як обрана функціональна модель, необхідно визначити її параметри. Цей процес називається параметричною ідентифікацією. Для параметричної ідентифікації до обчислювальної системи підключаються необхідні вимірювальні засоби.

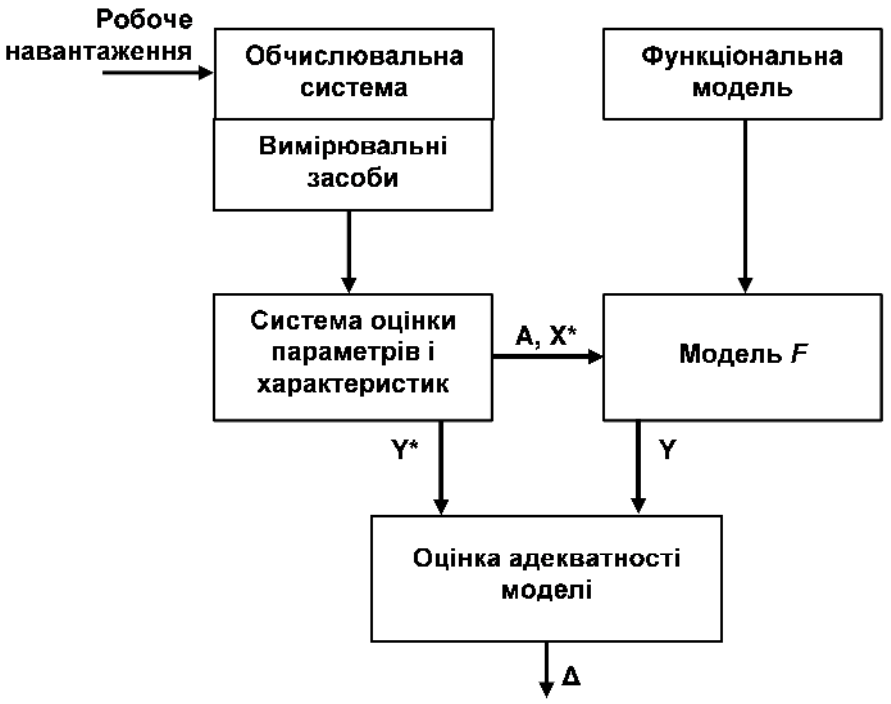


Рис. 1.14. Схема ідентифікації обчислювальної системи.

Одержувані дані використовуються системою оцінки параметрів і характеристик для обчислення параметрів X* і характеристик Y* системи, а також параметрів моделі A={a_n}. Система оцінки являє собою набір програм для обробки вимірювальних даних, що реалізує набір методів оцінки параметрів і характеристик. Обчислені значення параметрів A вводяться в модель, цілком визначаючи неї. Значення параметрів X* і характеристик Y* системи використовуються для перевірки адекватності моделі, тобто оцінки похибки Δ відтворення моделлю характеристик системи. Оцінка здійснюється шляхом порівняння значень характеристик *Y=F(X*)*, породжуваних моделлю, із зареєстрованими характеристиками Y* системи. Якщо модель адекватна системі, то використовується для прогнозування властивостей системи, що зводиться до обчислення на основі моделі характеристик *Y=F(X)*, що відповідають новим значенням X параметрів системи.

Задачі синтезу. Синтез — процес створення обчислювальної системи, що щонайкраще відповідає своєму призначенню. Вихідними в задачі синтезу є наступні дані, що характеризують призначення системи: 1) функція

системи (клас розв’язуваних задач); 2) обмеження на характеристики системи, наприклад на продуктивність, час відповіді, надійність і ін.; 3) критерій ефективності, що встановлює спосіб оцінки якості системи в цілому. Необхідно вибрати конфігурацію системи і режим обробки даних, що задовольняють заданим обмеженням і оптимальні за критерієм ефективності. Типова постановка задачі синтезу: спроектувати систему, що забезпечує розв’язок заданого класу задач А із продуктивністю не менш М задач на годину, середнім напрацюванням на відмову не менш T_0 і мінімальною вартістю.

Математично задача синтезу обчислювальної системи формулюється в такий спосіб. Нехай $\Theta=(\Theta_1,\ldots\Theta_Q)$ - вектор параметрів, що характеризують клас задач А, розв’язок яких є функцією системи; $S=(s_1,\ldots,s_p)$ - вектор параметрів, що характеризує конфігурацію (структуру) системи; $C = (c_1, \ldots, c_r)$ вектор параметрів режиму обробки; $Y=(y_1,\ldots, y_m)$ — вектор характеристик системи, зв’язаний з параметрами задач Θ, конфігурації S і режиму обробки C залежністю $Y=F(\Theta, S, C)$; $S=\{S_i\}$ - множина можливих конфігурацій обчислювальних систем; $C = \{C_j\}$ — множина можливих режимів обробки. Обмеження на характеристики і параметри системи $z_\alpha,\ldots, z_\omega \in (Y \cup X)$ будемо представляти у виді $z_\alpha \in z_\alpha^*,\ldots, z_\omega \in z_\omega^*$ де $z_\alpha^*,\ldots, z_\omega^*$ – області припустимих значень відповідних характеристик і параметрів. Критерій ефективності системи представляється заданою функцією $E = F(Y)$, що залежить від характеристик системи, що у свою чергу визначаються її параметрами $Y=F(\Theta, S, C)$. У встановлених позначеннях задача синтезу обчислювальної системи формулюється так: визначити конфігурацію 5 і режим обробки С, що максимізує ефективність системи

$$\max E = \max_{s\in S, c\in C} \Phi(Y) \qquad (1.2)$$

при виконанні обмежень:

$$z_\alpha \in z_\alpha^*,\ldots, z_\omega \in z_\omega^* \qquad (1.3)$$

На відміну від задачі аналізу, спрямованої на визначення характеристик системи Y по заданих параметрах X, завдання синтезу полягає у визначенні параметрів конфігурації S і режиму обробки C, що відповідають параметрам робочого навантаження Θ і характеристикам системи Y, заданим у виді (1.2), (1.3). Для задач синтезу характерні два наступних моменти. По-перше, передбачається наявність моделі $Y=F(\Theta, S, C)$, що встановлює залежність характеристик системи від її параметрів. По-друге, задача синтезу являє собою оптимізаційну задачу і припускає використання методу оптимізації, що відповідає виду цільової функції (1.2) і обмеженням (1.3). Метод оптимізації повинний гарантувати визначення глобального оптимуму цільової функції $E = F(Y)$, визначеної на множині конфігурацій S і режимів обробки C.

Складність задачі синтезу обчислювальної системи обумовлена числом варійованих параметрів, що описують конфігурацію і режим функціонування системи, і областю варіювання параметрів. При загальній постановці задачі синтезу, коли множині конфігурацій S і режимів обробки C містять у собі всі мислимі варіанти побудови систем (одномашинні і багатомашинні, мультіпроцесорні і мережні) і різні способи керування задачами, даними і завданнями, складність задачі синтезу перевершує можливості методів моделювання й оптимізації. Тому в загальній постановці задача синтезу обчислювальних систем виявляється нерозв’язною.

Для рішення задачі синтезу її спрощують поділом на послідовність етапів, на кожному з яких виявляються окремі аспекти організації системи.

1. Виходячи з призначення системи (клас розв’язуваних задач, технологія обробки даних, вимоги до продуктивності й умови роботи) і стану елементної бази визначається клас обчислювальної системи: одномашинна система, мультипроцессорний комплекс, локальна мережа й ін. При цьому аналізується ефективність систем різних класів і вибирається клас, що щонайкраще задовольняє призначенню системи.

2. В обраному класі систем синтезується архітектура системи:

визначається склад пристроїв, їхні функціональні можливості і технічні характеристики, типи інтерфейсів і структура зв’язків між пристроями, щонайкраще відповідають призначенню системи.

3. Визначається режим обробки даних і його параметри (склад і функції системних процесів, алгоритми розподілу ресурсів між завданнями і задачами, типи і діапазони пріоритетів і ін.). Цим установлюються функції керуючих програм операційної системи і склад системного програмного забезпечення.

Навіть при поділі задачі синтезу на три чи більші числа етапів синтез, зв’язаний з кожним етапом, не вдається звести до єдиної математичної процедури — задачі математичного програмування. Це обумовлено двома основними причинами. По-перше, синтез зв’язаний з різнотипними параметрами: одні є кількісними, а інші — якісними, тобто ознаками типу структури, пристроїв, пам’яті, інтерфейсів, способів керування процесами й ін. Тому синтез зводиться до задач чисельного математичного програмування, а також до комбінаторних задач на сполученнях, відносинах і т.д. Об’єднання різнотипних задач в одну задачу оптимізації виявляється не результативним через різнотипність обчислювальних процедур, що повинні використовуватися в процесі оптимізації. По-друге моделі, що маються в розпорядженні дослідників носять, як правило, локальний характер, відтворюючи властивості досить вузького класу структурних рішень і режимів функціонування. Об’єднання простих моделей у складну приводить до розривів функцій, не опуклим залежностям, що не тільки утрудняє, але практично виключає можливість застосування методів оптимізації.

З цих причин при синтезі систем прагнуть по можливості зменшувати розмірність задач шляхом поділу задачі синтезу на послідовність етапів, що зводяться до чисто комбінаторних, чи задачам чисельної оптимізації. При цьому проектування системи ведеться зверху вниз — від найбільш загальних рішень, зв’язаних із системою в цілому, до часткових рішень, що відноситься до окремих підсистем і їхніх частин. При розвязку багатьох задач синтезу

приходиться використовувати дуже прості моделі функціонування системи і її складових, що приблизно представляють залежності між характеристиками і параметрами. У цих умовах вибір проектних рішень здійснюється на основі досвіду й інтуїції розроблювачів. Таким чином, синтез обчислювальних систем зводиться до розв'язку значного числа взаємозалежних задач вибору способів організації і визначення параметрів проектованої системи в різних аспектах її організації й у відношенні до різних підсистем і елементів. При цьому використовуються як формальні, так і евристичні методи, причому на останні приходиться значне число проектних задач, що виходять за рамки можливостей відомих методів теорії обчислювальних систем.

Контрольні питання до розділу 1

1. Взаємозв'язок дисципліни комп'ютерні системи і паралельної обробки інформації.
2. Класифікація комп'ютерних систем по Флінну.
3. КС класу SIMD. Векторні і векторно-конвеєрні КС.
4. Підвищення продуктивності КС за рахунок векторної обробки даних. Структура векторного процесора.
5. Матричні обчислювальні системи. Загальна структура, переваги і недоліки.
6. Способи організації масивів процесорів в матричних обчислювальних системах.
7. Асоціативні обчислювальні системи.
8. Комп'ютерні системи з систолічною архітектурою.
9. Класифікація систолічних архітектур і їх топологія.
10. Комп'ютерні системи з командними словами надвеликої довжини (WLIV).
11. Комп'ютерні системи з явним паралелізмом команд (EPIC).
12. Паралельні комп'ютери із загальною пам'яттю. Переваги і недоліки.
13. Паралельні комп'ютери із розподіленою пам'яттю. Переваги і недоліки.
14. Що є предметом теорії обчислювальних систем?
15. В чому суть задач ідентифікаії систем?
15. В чому суть задач синтезу систем?

РОЗДІЛ 2 СУЧАСНІ ОБЧИСЛЮВАЛЬНІ СИТЕМИ СИСТЕМИ ТА ЇХ ТОПОЛОГІЇ

Тема 2.1 Мультикомп’ютерні КС

2.1.1 Комп’ютерні системи класу MIMD

Технологія SIMD історично почала освоюватися раніше, що і зумовило широке розповсюдження SIMD-систем. Проте в даний час намітився стійкий інтерес до архітектури класу MIMD. MIMD-системи володіють більшою гнучкістю, зокрема можуть працювати і як високопродуктивні однокористувацькі системи, і як багатопрограмні ОС, такі, що виконують безліч завдань паралельно. Крім того, архітектура MIMD дозволяє найефективніше розпорядитися всіма перевагами сучасної мікропроцесорної технології.

У MIMD-системі кожен процесорний елемент (ПЕ) виконує свою програму достатньо незалежно від інших ПЕ. В той же час ПЕ повинні якось взаємодіяти один з одним. Відмінність в способі такої взаємодії визначає умовне ділення MIMD-систем на ОС із загальною пам’яттю і системи з розподіленою пам’яттю. У *системах із загальною пам’яттю,* які характеризують як *сильно зв’язані* (tightly coupled), є загальна пам’ять даних і команд, доступна всім процесорним елементам за допомогою загальної шини або мережі з’єднань. До цього типу, зокрема, відносяться *симетричні мультипроцесори* (SMP, Symmetric Multiprocessor) і *системи з неоднорідним доступом до пам’яті* (NUMA, Non-Uniform Memory Access).

У *системах з розподіленою пам’яттю* або *слабо зв’язаних* (loosely coupled) багатопроцесорних системах вся пам’ять розподілена між процесорними елементами, і кожен блок пам’яті доступний тільки «своєму» процесору. Мережа з’єднань зв’язує процесорні елементи один з одним. Представниками цієї групи можуть служити *системи з масовим паралелізмом* (MPP, Massively Parallel Processing) і *кластерні обчислювальні системи.*

Базовою моделлю обчислень на MIMD-системі є сукупність незалежних процесів, що періодично звертаються до спільно використовуваних даних. Існує безліч варіантів цієї моделі. На одному кінці спектру — розподілені обчислення, в рамках яких програма ділиться на досить велике число паралельних завдань, що складаються з безлічі підпрограм.

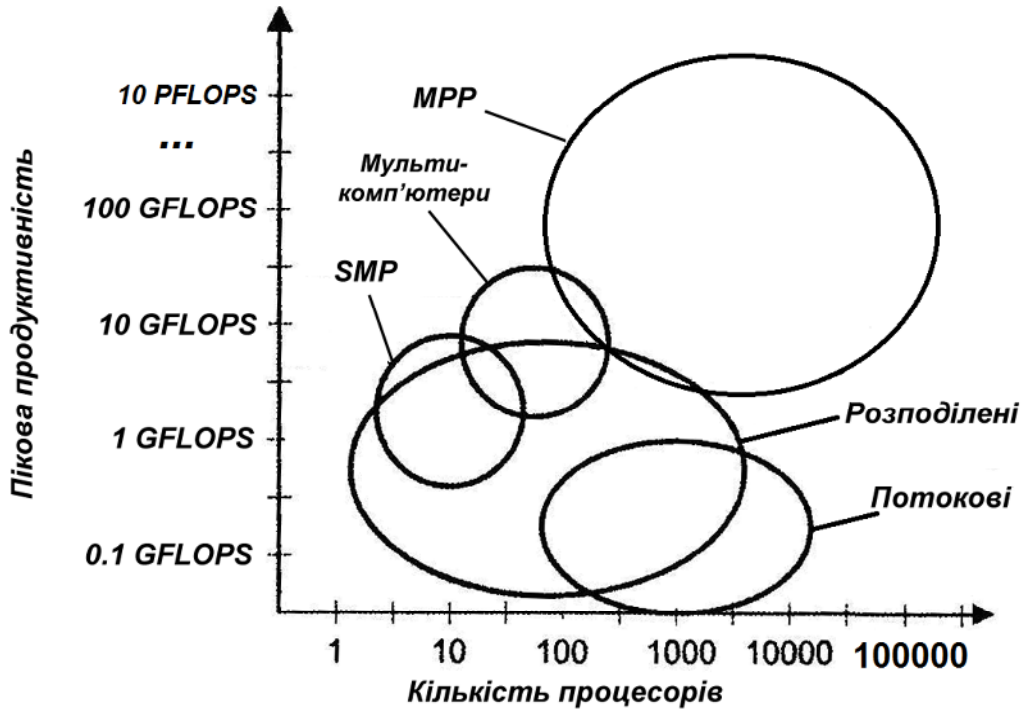


Рис. 2.1 Продуктивність MIMD-систем як функція їх типу і кількості процесорів

На іншому кінці — модель потокових обчислень, де кожна операція в програмі може розглядатися як окремий процес. Така операція чекає надходження вхідних даних (операндів), які повинні бути передані їй іншими процесами. Після їх отримання операція виконується, і результуюче значення передається тим процесам, які його потребують. Приблизні значення пікової продуктивності для різних типів систем класу MIMD показані на рис. 2.1.

2.1.2 Архітектури SMP

До порівняно недавнього часу практично всі однокористувацькі персональні системи і робочі станції містили по одному мікропроцесору

загального призначення. У міру зростання вимог до продуктивності і зниження вартості мікропроцесорів виробники обчислювальних засобів як альтернативу однопроцесорними ЕОМ почали пропонувати *симетричні мультипроцесорні обчислювальні системи*, так звані *SMP-системи* (SMP, Symmetric Multiprocessor). Це поняття відноситься як до архітектури ОС, так і до поведінки операційної системи, що відображає дану архітектурну організацію. SMP можна визначити як обчислювальну систему, що володіє наступними характеристиками:

- Є два або більш за процесори співставимої продуктивності;
- Процесори спільно використовують основну пам’ять і працюють в єдиному віртуальному і фізичному адресному просторі;
- Всі процесори зв’язані між собою за допомогою шини або по іншій схемі, так що час доступу до пам’яті будь-якого з них однаковий;
- Всі процесори розділяють доступ до пристроїв вводу/виводу або через одні і ті ж канали, або через різні канали, що забезпечують доступ до одного і того ж зовнішнього пристрою;
- Всі процесори здатні виконувати однакові функції (цим пояснюється термін «симетричні»).

Будь-який з процесорів може обслуговувати зовнішні переривання. Обчислювальна система управляється інтегрованою операційною системою, яка організовує і координує взаємодію між процесорами і програмами на рівні завдань, завдань, файлів і елементів даних. Звернемо увагу на останній пункт, який підкреслює одну з відмінностей по відношенню до слабо зв’язаних мультипроцесорних систем, таких як кластери, де як фізичною одиницею обміну інформацією зазвичай виступає повідомлення або цілий файл. У SMP допустима взаємодія на рівні окремого елементу даних, завдяки чому може бути досягнута високого ступеня зв’язності між процесами.

Хоча технічно SMP-системи симетричні, в їх роботі присутній невеликий чинник перекосу, який вносить програмне забезпечення. На час завантаження системи один з процесорів отримує статус ведучого (master).

Це не означає, що пізніше, під час роботи якісь процесори будуть веденими — всі вони в SMP-системі рівноправні. Термін «ведучий» вводиться тільки потім, щоб вказати який з процесорів по замовчуванню керуватиме первинним завантаженням ОС.

Операційна система планує процеси або нитки процесів (threads) відразу по всіх процесорах, приховуючи при цьому від користувача багатопроцесорний характер SMP-архітектури.

В порівнянні з однопроцесорними схемами SMP-системи мають переваги в наступних показниках:

Продуктивність. Якщо задача, що підлягає рішенню, піддається розбиттю на декілька частин так, що окремі частини можуть виконуватися паралельно, то безліч процесорів дає виграш в продуктивності відносно одиночного процесора того ж типу (рис. 2.2);

Готовність. У симетричному мультипроцесорі відмову одного з компонентів не веде до відмови системи, оскільки будь-який з процесорів в змозі виконувати ті ж функції, що і інші;

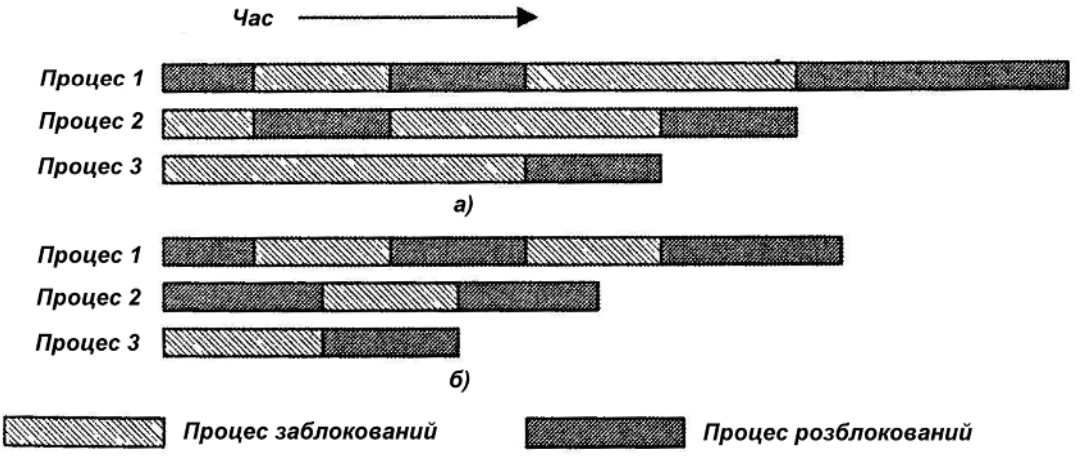


Рис. 2.2 Мультипрограмування і мультипроцесорна обробка
а — мультипрограмування; б — мультипроцесорна обробка.

Розширюваність. Продуктивність системи може бути збільшена шляхом додавання додаткових процесорів.

Масштабованість. Варіюючи число процесорів в системі, можна створити системи різної продуктивності і вартості.

Необхідно відзначити, що перераховане — це тільки потенційні переваги, реалізація яких неможлива, якщо в операційній системі відсутні засоби для підтримки паралелізму.

На рис. 2.3 в найзагальнішому вигляді показана архітектура симетричної мультипроцесорної ОС.

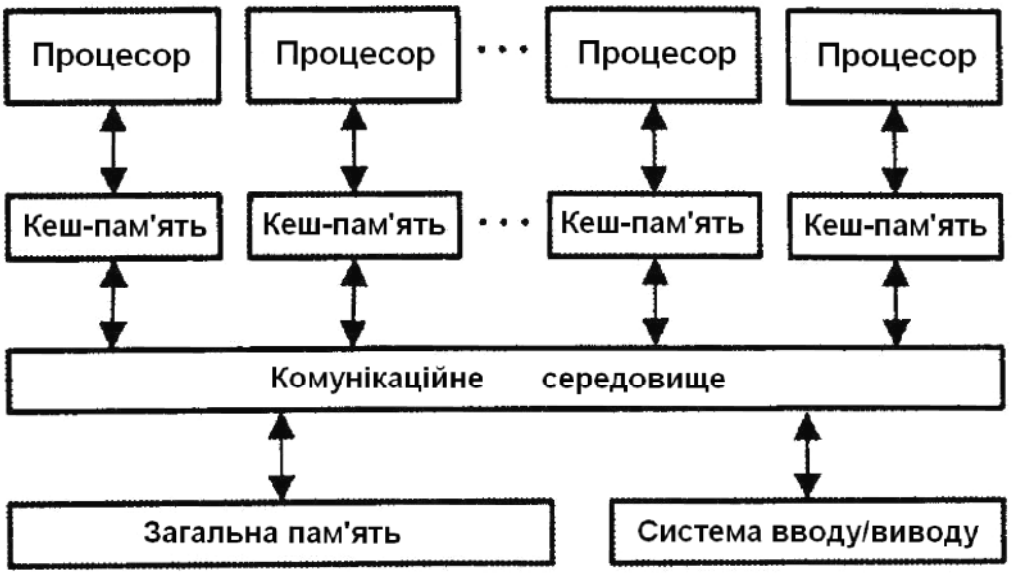


Рис. 2.3 Організація симетричної мультипроцесорної системи

Типова SMP-система містить від двох до 32 ідентичних процесорів, в якості яких зазвичай виступають недорогі RISC-процесори, такі, наприклад, як DEC Alpha, Sun SPARC, MIPS або HP PA-RISC. Останнім часом намітилася тенденція оснащення SMP-систем також і CISC-процесорами, зокрема виробників Intel та AMD.

Кожен процесор забезпечений локальною кеш-пам’яттю, що складається з кеш-пам’яті першого (L1) і другого (L2) рівнів. Узгодженість вмісту кеш-пам’яті всіх процесорів забезпечується апаратними засобами. У деяких SMP-системах проблема когерентності знімається за рахунок сумісного використовуваної кеш-пам’яті (рис. 2.4). На жаль, цей прийом технічно і економічно виправданий лише, якщо число процесорів не перевищує чотирьох. Застосування загальної кеш-пам’яті супроводжується підвищенням вартості і зниженням швидкодії кеш-пам’яті.

Всі процесори ОС мають рівноправний спільний доступ до основної пам’яті і пристроїв вводу/виводу. Така можливість забезпечується комунікаційною системою. Зазвичай процесори взаємодіють між собою через основну пам’ять (повідомлення і інформація про стан залишаються в області спільних даних). У деяких SMP-системах також передбачається прямий обмін сигналами між процесорами.

Пам’ять системи зазвичай будується за модульним принципом і організована так, що допускається одночасне звернення до її різних модулів (банків).

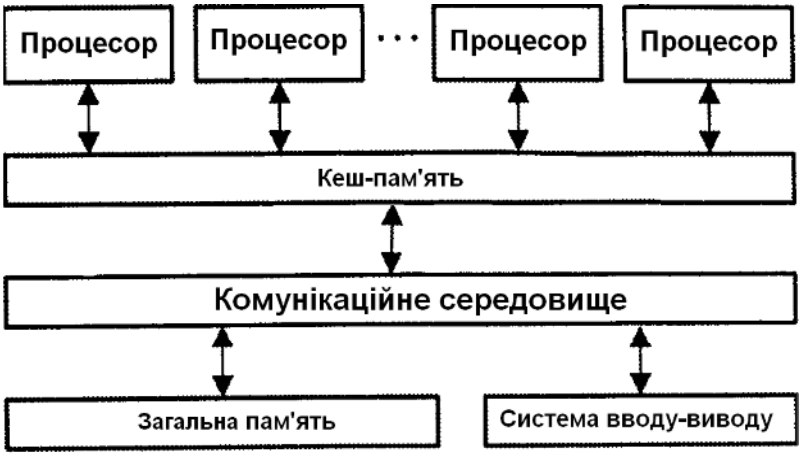


Рис. 2.4. SMP-система із спільно використовуваною кеш-пам’яттю

У деяких конфігураціях окрім спільно використовуваних ресурсів кожен процесор володіє також власними локальною основною пам’яттю і каналами вводу/виводу .

Важливим аспектом архітектури симетричних мультипроцесорів є спосіб взаємодії процесорів із загальними ресурсами (пам’яттю і системою вводу/виводу). З цих позицій можна виділити наступні види архітектури SMP-систем:

- із загальною шиною і часовим розділенням;
- з комутатором типу «кросбар»;
- з багатопортовою пам’яттю;
- з централізованим пристроєм управління.

Архітектура із загальною шиною

Структура і інтерфейси загальної шини в основному такі ж, як і в однопроцесорній ОС, де шина служить для внутрішніх з'єднань (рис. 2.5).

Рис. 2.5. Структура SMP-системи із загальною шиною

Переваги системи комунікації на базі загальної шини з розділенням часу – простота організації, низька вартість. До недоліків слід віднести можливість формування черг запитів, які викликають простій процесорів, складність взаємодії при великій кількості процесорів. Стосовно SMP-систем відзначимо, що фізичний інтерфейс, а також логіка адресації, арбітражу і розділення часу залишаються тими ж, як і в однопроцесорних системах.

Загальна шина дозволяє легко розширювати систему шляхом підключення до себе більшого числа процесорів. Крім того, нагадаємо, що шина — це, по суті пасивне середовище, і відмова одного з підключених до неї пристроїв не приводить до відмови всієї системи.

В той же час SMP-системам на базі загальної шини властива і основна властивість шинної організації — невисока продуктивність: швидкість системи обмежена часом циклу шини. З цієї причини кожен процесор має кеш-пам'ять, що істотно зменшує число звернень до шини. Наявність великого числа кешів породжує проблему їх когерентності, і це одна з основних причин, через яку системи на базі загальної шини зазвичай містять

не дуже багато процесорів. Так, в системах Compaq AlphaServer GS140 і 8400 використовується не більше 14 процесорів Alpha 21264. SMP-система HP N9000 в максимальному варіанті складається з 8 процесорів PA-8500, а системи SMP Thin Nodes для RS/6000 фірми IBM можуть включати від двох до чотирьох процесорів POWER PC 604.

Архітектура із загальною шиною широко поширена в SMP-системах, побудованих на мікропроцесорах x86. До цієї групи входять: DELL Power Edge, IBM Netfinity, HP NetServer.

Архітектура з комутатором типу «кросбар»

Архітектура з комутатором типу «кросбар» (рис. 2.6) орієнтована на модульну побудову загальної пам’яті і покликана вирішити проблему обмеженої пропускної спроможності систем із загальною шиною.

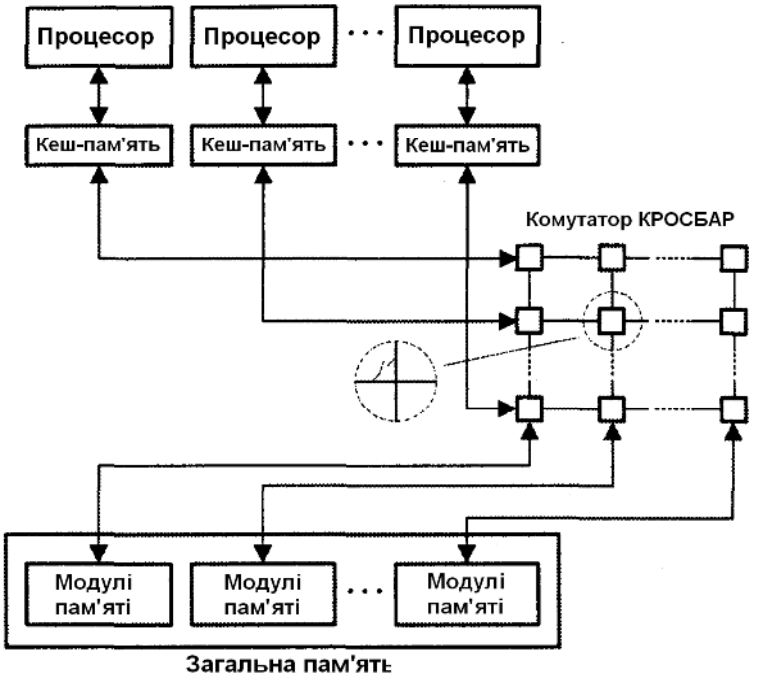


Рис. 2.6. Структура SMP-системи з комутатором типу «кросбар»

Комутатор забезпечує множинність шляхів між процесорами і банками пам’яті, причому топологія зв’язків може бути як двовимірною, так і тривимірною. Результатом стає вища смуга пропускання, що дозволяє будувати SMP-системи, що містять більше процесорів, ніж у випадку загальної шини. Типове число процесорів в SMP-системах на базі

матричного комутатора складає до 32 або 64. Відзначимо, що виграш в продуктивності досягається, лише коли різні процесори звертаються до різних банків пам’яті.

За логікою кросбара будується і взаємодія процесорів з пристроями вводу/виводу .

Як приклад ОС з розглянутою архітектурою можна навести систему Enterprise 10000, що складається з 64 процесорів, зв’язаних з пам’яттю за допомогою матричного комутатора GIGAPLANE-XB фірми Sun Microsystems (кросбар 16 х 16). У IBM RS/6000 Enterprise Server Model S70 комутатор типу «кросбар» забезпечує роботу 12 процесорів RS64. У SMP-системах ProLiant 8000 і 8500 фірм Compaq для об’єднання з пам’яттю і між собою восьми процесорів Pentium III Xeon застосована комбінація декількох шин і кросбара.

Концепція матричного комутатора (кросбара) не обмежується симетричними мультипроцесорами. Аналогічна структура зв’язків застосовується для об’єднання вузлів у ОС типу CC-NUMA і кластерних обчислювальних системах.

Архітектура з багатопортовою пам’яттю

Багатопортова організація запам’ятовуючго пристрою, забезпечує будь-якому процесору і модулю вводу/виводу прямий і безпосередній доступ до банків основної пам’яті (ОП). Такий підхід складніший, ніж при використанні шини, оскільки вимагає додавання до основної пам’яті додаткової, достатньо складної логіки. Проте це дозволяє підняти продуктивність, так як кожен процесор має виділений тракт до кожного модуля ОП. Інша перевага багатопортової організації — можливість призначити окремі модулі пам’яті як локальну пам’ять окремого процесора. Ця особливість дозволяє поліпшити захист даних від несанкціонованого доступу із сторони інших процесорів.

2.1.3 Кластерні обчислювальні системи

Один з найсучасніших напрямів в області створення обчислювальних систем — це *кластеризація*. По продуктивності і коефіцієнту готовності кластеризація є альтернативою симетричним мультипроцесорним системам. Поняття *кластер* визначимо як групу взаємно сполучених обчислювальних систем (вузлів), що працюють спільно, утворюючи єдиний обчислювальний ресурс і створюючи ілюзію наявності єдиної ОС. Вузлом кластера може виступати як однопроцесорна машина, так і ОС типу SMP або MPP. Важливо лише те, що кожен вузол в змозі функціонувати самостійно і окремо від кластера. У плані архітектури суть кластерних обчислень зводиться до об’єднання декількох вузлів високошвидкісною мережею. Для опису такого підходу, крім терміну «кластерні обчислення», достатньо часто застосовують такі назви, як: *кластер робочих станцій* (workstation cluster), *гіперобчислення* (hypercom-puting), *паралельні обчислення на базі мережі* (network-based concurrent computing), *ультраобчислення* (ultracomputing).

Спочатку перед кластерами ставилися два завдання: досягти великої обчислювальної потужності і забезпечити підвищену надійність ОС. Піонером в області кластерної архітектури вважається корпорація DEC, що створила перший комерційний кластер на початку 80-х років минулого століття.

Як вузли кластерів можуть використовуватися як однакові ОС (гомогенні кластери), так і різні (гетерогенні кластери). По своїй архітектурі кластерна ОС є слабо зв’язаною системою.

Переваги, що досягаються за допомогою кластеризації:

Абсолютна масштабованість. Можливе створення великих кластерів, що перевершують по обчислювальній потужності навіть найпродуктивніші одиночні машини. Кластер може містити десятки вузлів, кожен з яких є мультипроцесором.

Нарощувана масштабованість. Кластер будується так, що його

можна нарощувати, додаючи нові вузли невеликими порціями. Таким чином, користувач може почати з помірної системи, розширюючи її в міру необхідності.

Високий коефіцієнт готовності. Оскільки кожен вузол кластера — самостійна машина або ОС, відмова одного з вузлів не приводить до втрати працездатності кластера. У багатьох системах відмовостійкість автоматично підтримується програмним забезпеченням.

Добре співвідношення ціна/продуктивність. Кластер будь-якої продуктивності можна створити, з'єднуючи стандартні «будівельні блоки», при цьому його вартість буде нижча, ніж у одиночної машини з еквівалентною обчислювальною потужністю.

На рівні апаратного забезпечення кластер — це просто сукупність незалежних обчислювальних систем, об'єднаних мережею. При з'єднанні машин в кластер майже завжди підтримуються прямі міжмашинні зв'язки. Рішення можуть бути простими, такими, що грунтуються на апаратурі Ethernet, або складними з високошвидкісними мережами з пропускною спроможністю в сотні мегабайтів в секунду. До останньої категорії відносяться RS/6000 SP компанії IBM, системи фірми Digital на основі Memory Channel, ServerNet корпорації Compaq.

Вузли кластера контролюють працездатність один одного і обмінюються специфічною, характерною для кластера інформацією. Контроль працездатності здійснюється за допомогою спеціального сигналу, часто званого *heartbeat*, що можна перевести як «серцебиття». Цей сигнал передається вузлами кластера один одному, щоб підтвердити їх нормальне функціонування.

Невід'ємна частина кластера — спеціалізоване програмне забезпечення (ПЗ), на яке покладається завдання забезпечення безперебійної роботи при відмові одного або декількох вузлів. Таке ПЗ проводить перерозподіл обчислювального навантаження при відмові одного або декількох вузлів кластера, а також відновлення обчислень при збої у вузлі. Крім того, за

наявності в кластері спільно використовуваних дисків кластерне ПЗ підтримує єдину файлову систему.

Класифікація архітектури кластерних систем

У літературі приводяться різні способи класифікації кластерів. Так, в простому варіанті орієнтуються на те, чи є диски в кластері що розділяються всіма вузлами. На рис. 2.7, *а* показаний кластер з двох вузлів, спільна робота яких координується за рахунок високошвидкісної лінії, по якій відбувається обмін повідомленнями. Такою лінією може бути локальна мережа, що може використовується також і комп'ютерами які не входять в кластер, або виділена лінія.

У останньому випадку один або декілька вузлів кластера матимуть вихід в локальну або глобальну мережу, завдяки чому забезпечується зв'язок між серверним кластером і видаленими клієнтськими системами.

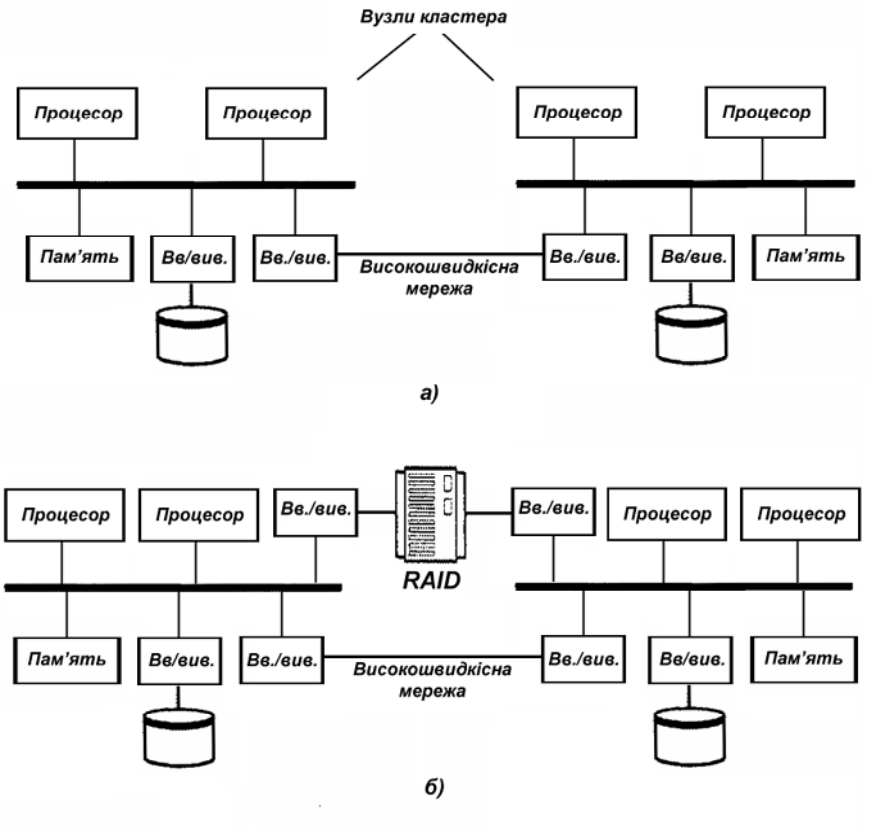


Рис. 2.7 Конфігурації кластерів:
***а* - без спільно використовуваних дисків;**
***б* - із спільно використовуваними дисками.**

Яснішу картину дає класифікація кластерів на основі схожості їх функціональних особливостей. Така класифікація приведена в табл. 2.1.

Таблиця 2.1. Методи кластеризації

Метод кластеризації	Опис
Пасивне резервування	Вторинний сервер при відмові первинного бере управління на себе.
Резервування з активним вторинним сервером	Вторинний сервер, як і первинний, використовується при вирішенні завдань.
Самостійні сервери	Самостійні сервери мають власні диски, а дані постійно копіюються з первинного сервера на вторинний
Сервери з підключенням до всіх дисків	Сервери підключені до одних і тих же дисків, але кожен сервер володіє своєю влачною частиною. Якщо один з серверів відмовляє, то управління його дисками бере на себе інший сервер
Сервери із спільно використовуваними дисками	Безліч серверів працюють в режимі колективного доступу до дисків

Кластеризація з резервуванням — найбільш старий і універсальний метод. Один з серверів бере на себе все обчислювальне навантаження, тоді як другий залишається неактивним, але готовим перейняти обчислення при відмові основного сервера. Активний або первинний сервер періодично посилає резервному тактуюче повідомлення. За відсутності тактуючих повідомлень (це розглядається як відмова первинного сервера) вторинний сервер бере управління на себе. Такий підхід підвищує коефіцієнт готовності, але не покращує продуктивності. Більш того, якщо єдиний вид взаємодії між вузлами — обмін повідомленнями, і якщо обидва сервери кластера не використовують диски колективно, то резервний сервер не має доступу до баз даних, керованих первинним сервером.

Пасивне резервування для кластерів не характерне. Термін «кластер» відносять до безлічі взаємозв’язаних вузлів, що беруть активну участь в обчислювальному процесі і спільно створюють ілюзію однієї потужної

обчислювальної машини. До такої конфігурації зазвичай застосовують поняття системи *з активним вторинним сервером,* і тут виділяють три методи кластеризації: самостійні сервери, сервери без сумісного використання дисків і сервери з сумісним використанням дисків.

При першому підході кожен вузол кластера розглядається як самостійний сервер з власними дисками, причому жоден з дисків в системі не є загальним (див. рис. 2.7, *а).* Схема забезпечує високу продуктивність і високий коефіцієнт готовності, проте вимагає спеціального програмного забезпечення для планування розподілу клієнтських запитів по серверах так, щоб добитися збалансованого і ефективного навантаження на кожен з них. Необхідно також створити умови, щоб при відмові одного з вузлів в процесі виконання якого-небудь завдання інший вузол міг перехопити і завершити задачу, що залишилася без управління. Для цього дані в кластері повинні постійно копіюватися, щоб кожен сервер мав доступ до всієї найбільш свіжої інформації в системі. Через ці витрати високий коефіцієнт готовності досягається лише за рахунок втрати продуктивності.

Для скорочення комунікаційних витрат більшість кластерів в даний час формують з серверів, підключених до загальних дисків, зазвичай представлених дисковим масивом RAID (рис. 2.7, *б*).

Один з варіантів такого підходу припускає, що сумісний доступ до дисків не застосовується. Загальні диски розбиваються на розділи, і кожному вузлу кластера виділяється свій розділ. Якщо один з вузлів відмовляє, кластер може бути реконфігурований так, що права доступу до його частини загального диска передаються іншому вузлу.

У другому варіанті безліч серверів розділяють в часі доступ до загальних дисків, так що будь-який вузол має можливість звернутися до будь-якого розділу кожного загального диска. Ця організація вимагає наявність яких-небудь засобів блокування, що гарантують, що у будь-який момент часу доступ до даних матиме тільки один з серверів.

Обчислювальні машини (системи) в кластері взаємодіють відповідно

до одному із двох транспортних протоколів. Перший з них, протокол TCP (Transmission Control Protocol), оперує потоками байтів, гарантуючи надійність доставки повідомлення, другий - UDP (User Datagram Protocol) намагається посилати пакети даних без гарантії їх доставки. Останнім часом застосовують спеціальні протоколи, які працюють набагато краще. Так, очолюваний компанією Intel консорціум (Microsoft, Compaq і ін.) запропонував новий протокол для внутрішньокластерних комунікацій, який називається Virtual Interface Architecture (VIA) і претендує на роль стандарту.

При обміні інформацією використовуються два програмні методи: *передачі повідомлень* і *розподіленої спільно використовуваної пам'яті*. Перший спирається на явну передачу інформаційних повідомлень між вузлами кластера. У альтернативному варіанті також відбувається пересилка повідомлень, але рух даних між вузлами кластера приховано від програміста і забезпечується апаратно.

Кластери забезпечують високий рівень доступності — в них відсутні єдина операційна система і спільно використовувана пам'ять, тобто немає проблеми когерентності кешів. Крім того, спеціальне програмне забезпечення в кожному вузлі постійно контролює працездатність решти всіх вузлів. Цей контроль заснований на періодичній розсилці кожним вузлом сигналу «Поки живий» (keepalive). Якщо сигнал від деякого вузла не поступає, то такий вузол вважається таким, що вийшов з ладу; йому не дається можливість виконувати ввід/вивід, його диски і інші ресурси (включаючи мережеві адреси) перепризначувалися іншим вузлам, а програми, що виконувалися ним, перезапускаються на інших вузлах.

Кластери добре масштабуються в плані продуктивності при додаванні вузлів. У кластері може виконуватися декілька окремих завдань, але для масштабування окремої задачі потрібно, щоб її частини погоджували свою роботу шляхом обміну повідомленнями. Проте, не можна не враховувати, що взаємодії між вузлами кластера займають значно більше часу, чим в традиційних ОС.

Можливість практично необмеженого нарощування числа вузлів і відсутність єдиної операційної системи робить кластерну архітектуру виключно успішно масштабованою, і навіть системи з сотнями і тисячами вузлів показують себе на практиці з позитивного боку.

Топології кластерів

При створенні кластерів з великою кількістю вузлів можуть застосовуватися найрізноманітніші топології. У даному розділі зупинимося на тих, які характерні для найбільш поширених «малих» кластерів, що складаються з 2-4 вузлів.

Топологія кластерних пар

Топологія кластерних пар знаходить застосування при організації двох або чотирьохвузлових кластерів (рис. 2.8).

Вузли групуються попарно. Дискові масиви приєднуються до обох вузлів пари, причому кожен вузол має доступ до всіх дискових масивів своєї пари. Один з вузлів є резервним для іншого.

Чотирьохвузлова кластерна «пара» є простим розширенням двохвузлової топології. Обидві кластерні пари з погляду адміністрування і настройки розглядаються як єдине ціле.

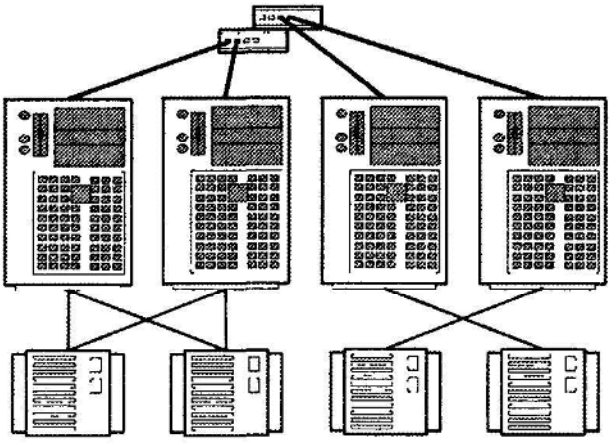


Рис. 2.8 Топологія кластерних пар

Ця топологія підходить для організації кластерів з високою готовністю даних, але відмовостійка реалізується тільки в межах пари, оскільки пристрої

зберігання інформації, що належать нею, не мають фізичного з'єднання з іншою парою. Приклад: організація паралельної роботи СУБД Informix XPS.

Топологія N + 1

Топологія *N*+1 дозволяє створювати кластери з 2,3 і 4 вузлів (рис. 2.9).

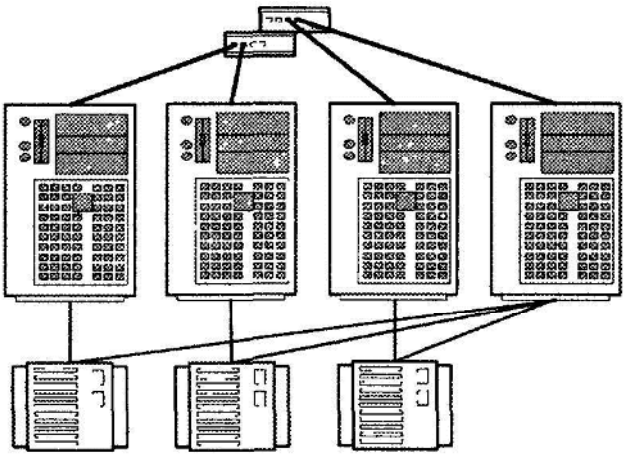


Рис. 2.9 Топологія N + 1

Кожен дисковий масив підключаються тільки до двох вузлів кластера. Дискові масиви організовані по схемі RAID 1 (детальніше розглянуто в розділі 3). Один сервер має з'єднання зі всіма дисковими масивами і служить як резервний для решти (основних або активних) всіх вузлів. Резервний сервер може використовуватися для підтримки високого ступеня готовності в парі з будь-яким з активних вузлів.

Топологія рекомендується для організації кластерів високої готовності. У тих конфігураціях, де є можливість виділити один вузол для резервування, ця топологія сприяє зменшенню навантаження на активні вузли і гарантує, що навантаження вузла, що вийшов з ладу, буде відтворено на резервному вузлі без втрати продуктивності. Відмовостійкість забезпечується між будь-яким з основних вузлів і резервним вузлом. В той же час топологія не дозволяє реалізувати глобальну відмовостійкість, оскільки основні вузли кластера і їх системи зберігання інформації не зв'язані один з одним.

Топологія N x N

Аналогічно топології *N*+1, топологія *N x N* (рис.2.10) розрахована на

створення кластерів із 2, 3 і 4 вузлів, але на відміну від першої володіє більшою гнучкістю і масштабованістю.

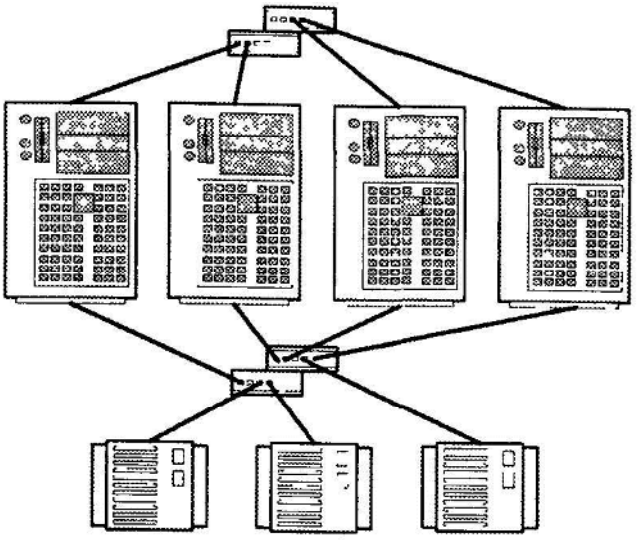


Рис. 2.10 Топологія N x N

Тільки у цій топології всі вузли кластера мають доступ до всіх дискових масивів, які, у свою чергу, будуються за схемою RAID 1 (з дублюванням). Масштабованість виявляється в простоті додавання до кластера додаткових вузлів і дискових масивів без зміни з'єднань в існуючій системі.

Топологія дозволяє організувати каскадну систему відмовостійкої, при якій обробка переноситься з несправного вузла на резервний, а у разі його виходу з ладу — на наступний резервний вузол і т. д. Кластери з топологією *N x N* забезпечують підтримку застосування Oracle Parallel Server, що вимагає з'єднання всіх вузлів зі всіма системами зберігання інформації. В цілому топологія характеризується кращою відмовостійкою і гнучкістю в порівнянні з іншими рішеннями.

Топологія з повністю роздільним доступом

У топології з повністю роздільним доступом (рис. 2.11) кожен дисковий масив з'єднується тільки з одним вузлом кластера.

Топологія рекомендується тільки для тих застосувань, для яких

характерна архітектура повністю роздільного доступу, наприклад для вже згадуваної СУБД Informix XPS.

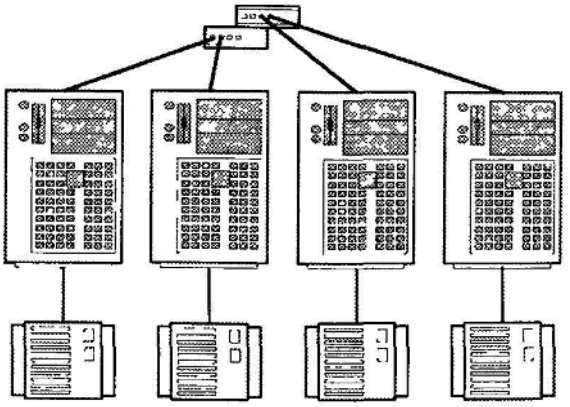


Рис. 2.11 Топологія з повністю роздільним доступом

2.1.4 Системи з масовою паралельною обробкою (MPP)

Основною ознакою, по якій обчислювальну систему відносять до *архітектури з масовою паралельною обробкою* (MPP, Massively Parallel Processing), служить кількість процесорів *n*. Строгої межі не існує, але зазвичай вважається що при *n* > 32 це вже MPP система. Узагальнена структура MPP-системи показана на рис. 2.12.

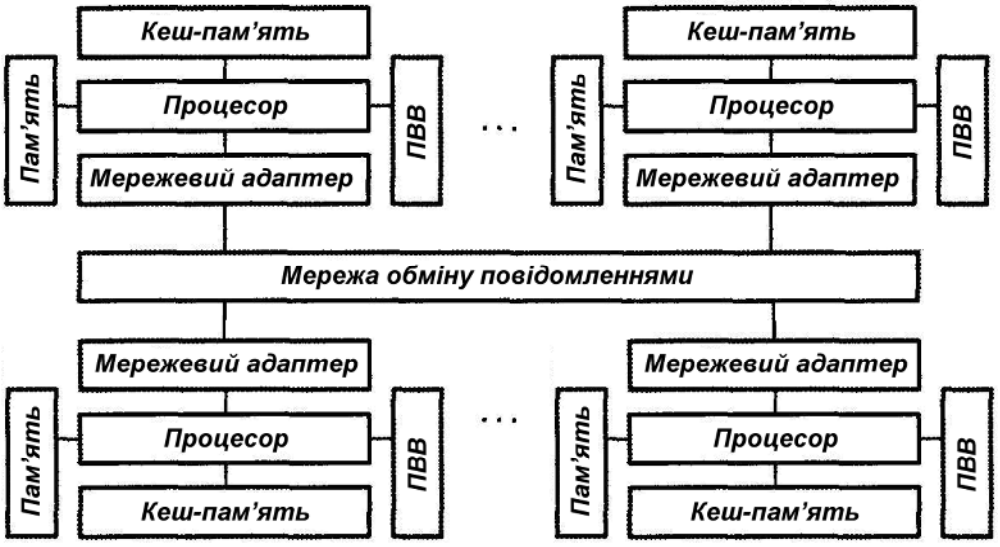


Рис. 2.12 Структура обчислювальної системи з масовою паралельною обробкою.

Головні особливості, по яких обчислювальну систему зараховують до класу MPP, можна сформулювати таким чином:

- стандартні мікропроцесори; фізично розподілена пам’ять;
- мережа з’єднань з високою пропускною спроможністю і малими затримками;
- хороша масштабованість (до десятків тисяч процесорів);
- асинхронна MIMD-система з пересилкою повідомлень;
- програма є безліччю процесів, що мають окремі адресні простори.

Основні причини появи систем з масовою паралельною обробкою — це, по-перше, необхідність побудови ОС з гігантською продуктивністю і, по-друге, прагнення розширити межі виробництва ОС у великому діапазоні, як у плані продуктивності, так і вартості. Для MPP-системи, в якій кількість процесорів може мінятися в широких межах, завжди реально підібрати конфігурацію із заздалегідь заданою обчислювальною потужністю і фінансовими затратами

Якщо говорити про MPP як про представника класу MIMD з розподіленою пам’яттю і відвернутися від організації вводу/виводу , то ця архітектура є природним розширенням кластерів на велике число вузлів. Звідси для MPP-систем характерні всі переваги і недоліки кластерів, причому у зв’язку з підвищеним числом процесорних вузлів як плюси, так і мінуси стають набагато вагомішими.

Характерна влістивість MPP-систем — наявність єдиного керуючого пристрою (процесора), який розподіляє завдання між безліччю підлеглих йому процесорів, найчастіше однакових (взаємозамінних), що належать одному або декільком класам. Схема взаємодії у загальних рисах досить проста:

- центральний керуючий пристрій формує чергу завдань, кожному з яких призначається деякий рівень пріоритету;
- у міру звільнення підлеглих процесорів їм передаються завдання з черги;

- підлеглі процесори оповіщають центральний процесор про хід виконання завдання, зокрема про завершення виконання або про потребу в додаткових ресурсах;
- у центрального процесора є засоби для контролю роботи підлеглих процесорів, зокрема для виявлення нештатних ситуацій, переривання виконання завдання у разі появи пріоритетнішого завдання і тому подібне;

У деякому наближенні можна вважати, що на центральному процесорі виконується ядро операційної системи (розподіл завдань), а на підлеглих — самі завдання. Підлеглість між процесорами може бути реалізована як на апаратному, так і на програмному рівні.

Зовсім не обов’язково, щоб MPP-система мала розподілену оперативну пам’ять, коли кожен процесорний вузол володіє власною локальною пам’яттю. Так, наприклад, системи SPP1000/XA і SPP1200/XA являють собою приклад ОС з масовим паралелізмом, пам’ять яких фізично розподілена між вузлами, але логічно вона загальна для всієї обчислювальної системи. Проте більшість MPP-систем мають як логічно, так і фізично розподілену пам’ять.

Завдяки властивості масштабованості, MPP-системи є сьогодні лідерами по досягнутій продуктивності; найбільш яскравий приклад цьому - суперкомп’ютер Cray Jaguar, що містить 224162 обчислювальні ядра, на базі 6-ти ядерних процесорів Opteron, пікова продуктивність становить 1,75 петафлопс (10^{15} операцій з дійсними числами в секунду). Іншим представником суперкомп’ютерів є IBM Roadrunner на базі 12960 дев’ятиядерних процесорів PowerXCell 8i з тактовою частотою частотою 3.2 ГГц, загальна пікова продуктивність становить з продуктивністю 1,04 петафлопс.

З іншого боку, розпаралелювання в MPP-системах в порівнянні з кластерами, що містять небагато процесорів, стає ще важчим завданням. Слід пам’ятати, що приріст продуктивності при зростанні числа процесорів

зазвичай взагалі досить швидко зменшується (див. закон Амдала). Крім того, достатньо важко знайти завдання, які зуміли б ефективно завантажити безліч процесорних вузлів. Сьогодні не так вже багато задач можуть ефективно виконуватися на МРР-системі. Має місце також проблема переносимості програм між системами з різною архітектурою. Ефективність розпаралелювання у багатьох випадках сильно залежить від деталей архітектури МРР-системи, наприклад топології з'єднання процесорних вузлів.

Найефективнішою була б топологія, в якій будь-який вузол міг би безпосередньо зв'язатися з будь-яким іншим вузлом, але у ОС на основі МРР це технічно важко реалізовується. Як правило, процесорні вузли в сучасних МРР комп'ютерах утворюють або двомірні і тримірні решітки (наприклад, в SNI/Pyramid RM1000**)** або гіперкуб (як в суперкомп'ютерах nCube).

Оскільки для синхронізації процесів, що паралельно виконуються, необхідний обмін повідомленнями, які повинні доходити з будь-якого вузла системи в будь-який інший вузол, важливою характеристикою є діаметр системи *D.* У разі двомірних решіток $D \sim 2(m-1)$, де m – кількість вузлів в одному вимірі квадратної решітки, у разі гіперкуба $D \sim n$, де n – це порядок гіперкуба. Таким чином, при збільшенні числа вузлів вигідніша архітектура гіперкуба.

Час передачі інформації від вузла до вузла залежить від стартової затримки і швидкості передачі. У будь-якому випадку, за час передачі процесорні вузли встигають виконати багато команд, і це співвідношення швидкодії процесорних вузлів і передавальної системи, ймовірно, зберігатиметься — прогрес в продуктивності процесорів набагато вагоміший, ніж в пропускній спроможності каналів зв'язку. Тому інфраструктура каналів зв'язку в МРР-системах є об'єктом найбільш пильної уваги розробників.

Слабким місцем МРР було і є центральний пристрій керування (ЦПК), — при виході його з ладу вся система виявляється непрацездатною. В основі підвищення надійності ЦПК лежить на шляхах спрощення його апаратури

і/або її дублювання.

Не дивлячись на всі складнощі, сфера застосування ОС з масовим паралелізмом постійно розширюється. Різні системи цього класу експлуатуються в багатьох ведучих суперкомп'ютерних центрах світу. Слід особливо відзначити комп'ютери Cray XT5 і Cray XT6, які ілюструють той факт, що світовий лідер виробництва векторних СУПЕРЕОМ, компанія Cray Research, вже не орієнтується виключно на векторні системи.

В якості прикладу сучасної MPP системи наведемо опис суперкомп'ютера Roadrunner, який став першим комп'ютером, що подолав на тесті Linpack рубіж продуктивності в 1 PFlop/s. Починаючи з 31-ої редакції (червень 2008 року), очолює список TOP500 найбільш потужіх комп'ютерів світу. Суперкомп'ютер створений компанією IBM для Міністерства Енергетики США і встановлений в Лос-аламоській національній лабораторії в Нью-Мексіко, США.

Суперкомп'ютер Roadrunner побудований по гібридній схемі з 6120 двоядерних процесорів AMD Opteron, що працюють на частоті 1.8 Ггц, і майже 12240 процесорів IBM Cell 8i, що працюють на частоті 3.2 Ггц, в спеціальних блейд-модулях TriBlades, сполучених за допомогою комунікаційної мережі Infiniband. Установка займає площу приблизно 560 квадратних метрів, і важить 226 тонн. Загальне енергоспоживання установки - 2.35 Мвт, при цьому енергоефективність складає 437 MFlop/s/Вт. Вартість IBM Roadrunner склала 133 мільйони доларів. Пікова продуктивність суперкомп'ютера склала 1.376 PFlop/s, продуктивність на тесті Linpack - 1.026 PFlop/s.

Логічно блейд-модуль TriBlade складається з чотирьох ядер Opteron, чотири PowerXCell 8i процесорів, 16 Гбайт пам'яті для Opteron і 16 Гбайт пам'яті для Cell. Фізично TriBlade складається з однієї плати LS21, плати розширення і двох плат QS22. LS21 містить два двоядерні процесори Opteron з 16 Гбайт пам'яті, по 4 Гбайт на ядро. Кожна плата QS22 містить два процесори PowerXCell 8i і 8 Гбайт пам'яті, по 4 Гбайт на кожен процесор.

Плата розширення сполучає QS22 через чотири роз’єми PCI x8 з LS21, по два роз’єми на QS22. Також вона забезпечує підключення Infiniband 4x DDR. В результаті один блейд-модуль TriBlade займає чотири слоти, і три TriBlades поміщаються в шасі BladeCenter H.

Система містить 49 Тбайт пам’яті для процесорів Cell та 49 Тбайт пам’яті для процесорів Opteron.

Roadrunner працює під управлінням Red Hat Enterprise Linux і управляється за допомогою програмного забезпечення xCAT.

Міністерство Енергетики планує використовувати RoadRunner для розрахунку старіння ядерних матеріалів і аналізу безбеки та надійності ядерного арсеналу США. Також планується використання для наукових, фінансових, транспортних і аерокосмічних розрахунків

Іншим яскравим представником суперкомп’ютерів є сімейство Cray XT53-XT6.

Cray XT3 з одноядерними Opteron з’явився незабаром після виходу цих процесорів на ринок в 2005 році, потім з’явилися конфігурації з двоядерними процесорами. XT4 на базі двоядерних Opteron був випущений в 2007 році. XT5 з чотириядерними процесорами Barcelona/2,6 Ггц (а потім і Shanghai/2,7 Ггц) і шестидерними Istanbul/2,6 Ггц, поставлявся вже у 2008 році. При цьому число процесорних роз’ємів у вузлі XT5 зросло до двох. Недавно в Cray оголосили про випуск систем Cray XT6 на базі шести- і восьмиядерних Opteron серії 6100.

Характеризуючи MPP-архітектуру Cray XT в цілому, слід підкреслити, що вона включає вузли, які зв’язані «фірмовим» комутаторами SeaStar з топологією тривимірного тора. Зупинимося на розгляді систем Cray XT5 і XT6. Вузли XT5 містять ASIC-мікросхему, що забезпечує роботу з міжз’єднання SeaStar2+ і два процесорні роз’єми для Opteron і слотів для модулів DIMM (рис.2.13). У Cray XT4 вузол був влаштований аналогічним чином, але мав один процесорний роз’єм.

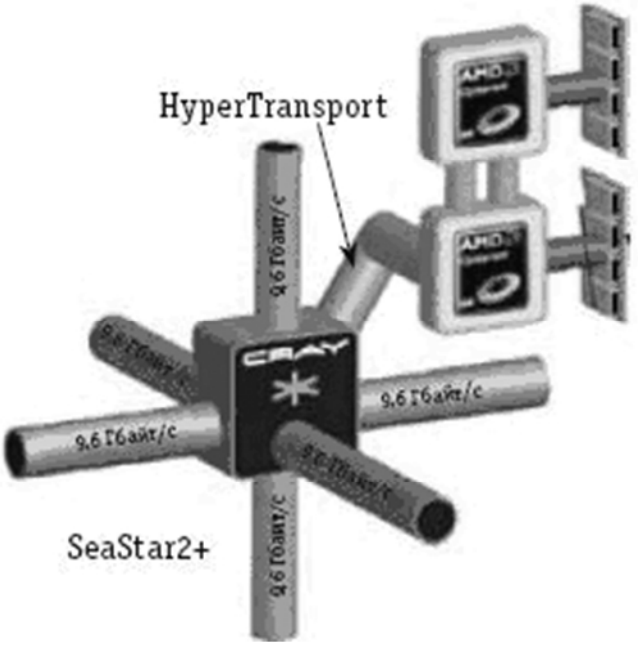


Рис. 2.13 Структура з’єднань через SeaStar2+

Для зв’язку мікросхеми комутатора з Opteron природним чином застосовуються інтегровані в мікропроцесор інтерфейси каналів HyperTransport 2.0 (пряме під’єднання оперативної пам’яті до процесора – одна з основних особливостей всіх моделей Opteron). В результаті пропускна спроможність оперативної пам’яті масштабується з числом процесорів, і з розрахунку на вузол пропускна спроможність пам’яті складає 25,6 Гбайт/с (застосовується захищена кодами ECC регістрова пам’ять DDR2-800). Об’єм оперативної пам’яті вузла складає від 8 Гбайт до 32 Гбайт .

В залежності від типу процесора що використовується в ситемі, пікова продуктивність вузла лежить в діапазоні приблизно від 70 GFLOPS до 124 GFLOPS.

Наступний рівень конструктиву над вузлом XT5 – лезо (Blaid), що містить чотири вузли. При використанні Istanbul продуктивність леза досягає 500 GFLOPS; йому відповідають грати вузлів 1x2x2.

У стійці можна реалізувати решітку вузлів 1x4x24 (24 леза по чотири вузли) з продуктивністю до 12 TFLOPS при ємкості оперативної пам’яті 1,54 Тбайт (з розрахунку 16 Гбайт на вузол), а в системі XT5 в цілому – решітку 25x32x24. Такій системі відповідає продуктивність порядка 2 PFLOPS і

ємкість пам’яті 300 Тбайт. Відповідно до вказаних на сайті Cray специфікаціями XT5, стійка займає площу приблизно 0,6м х 1,4 м при висоті близько 2 м, важить порядка 700 кг і споживає не більше 43 кВт.

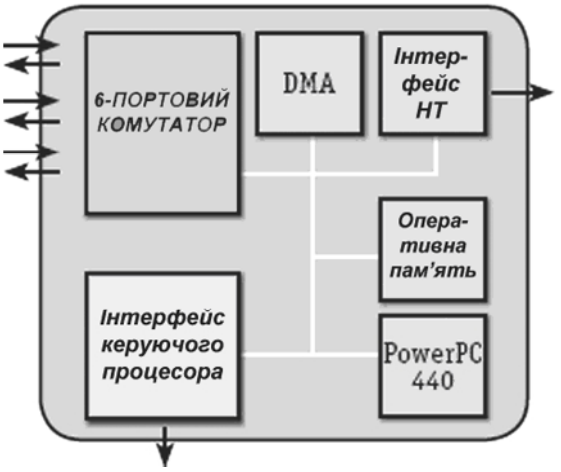


Рис.2.14 Структура комутатора SeaStar2+

Комутатор SeaStar2+ підтримується завдяки відповідній мікросхемі (Рис.2.14), що забезпечує маршрутизацію і що має шість портів з пропускною спроможністю 9,6 Гбайт/с на порт. Це відповідає сумарній пропускній спроможності мікросхеми в 57,6 Гбайт/с, тоді як затримка передачі з вузла у вузол не перевищує 2 мкс. Окрім шестипортового маршрутизатора і інтерфейсу HyperTransport, мікросхема SeaStar2+ містить механізм прямого доступу в пам’ять (Direct Memory Access, DMA), підтримку інтерфейсу управління лезом і інші засоби.

Модуль SeaStar2+ реалізований як знімний, що дозволяє здійснювати модернізацію комутатора окремо від процесорів і пам’яті. Тому Cray XT4 можна модернізувати до XT5, Cray XT5 – до XT6 (у останньому випадку міняються процесори і оперативна пам’ять), а в Xt6 комутатор, ймовірно, можна буде замінити на модуль на базі перспективної комунікаційної мікросхеми Cray Gemini, що відповідатиме переходу до систем, що здобули популярність під кодовою назвою Baker. Пізніше плануєтьсямодернізація з переходом на процесори AMD Bulldozer з 12 або 16 ядрами, які виготовлятимуться за 32-нанометровой технологією.

У тривимірній решітці частина вузлів може бути виділена як сервісні вузли (Рис.2.15), які мають один, а не два процесорні роз’єми, а для зв’язку з «зовнішнім світом» можуть застосовуватися шини PCI-Express. Додатково можуть використовуватися і мережеві протоколи Gigabit Ethernet, 10 Gigabit Ethernet, Infiniband, а також Fibre Channel. Можливе підключення дискових масивів Fibre Channel і SATA.

Таке підключення жорстких дисків характерне, коли застосовуються одиночні суперкомп’ютери Cray XT. Вузли, через які відбувається таке під’єднання, Cray називає SIO-узлами (Storage-I/O).

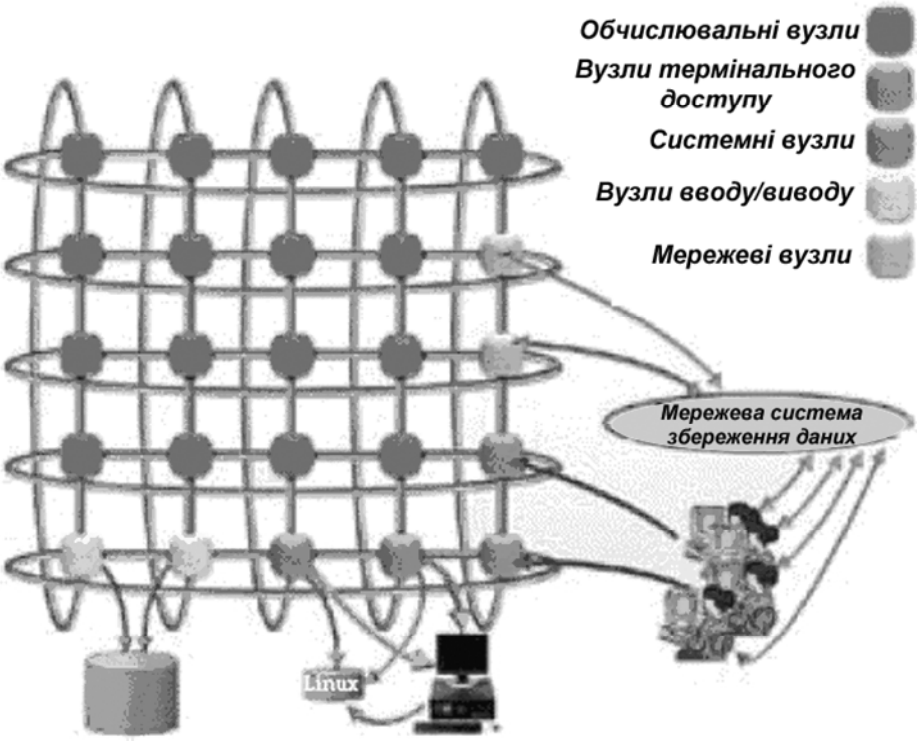


Рис.2.15 Топологія з’єднань вузлів у системі Cray XT

У випадку потреби використання спільних даних багатьма вузлами багатопроцесорної системи зазвичай застосовуються мережеві системи зберігання інформації. Cray пропонує високопродуктивні системи зберігання на базі Lustre - масштабованої, надійної паралельної файлової системи з відкритим кодом; вона забезпечує роботу тисяч вузлів, 1 Пбайт дискової пам’яті і пропускну спроможність в сотні гігабайт в секунду. Як партнер Cray в області систем зберігання виступає Sun Microsystems.

2.1.5 Обчислювальні системи з неоднорідним доступом до пам’яті

Основні платформи, які часто використовуються при створенні комерційних мультипроцесорних систем - це SMP, MPP і кластери. Разом з цим останнім часом почали з’являтися рішення, в яких акцентується спосіб організації пам’яті. Мова йде про ОС, побудованих відповідно до технології неоднорідного доступу до пам’яті (NUMA, Non-Uniform Memory Access), точніше з кеш-когерентним доступом до неоднорідної пам’яті (CC-NUMA).

У симетричних мультипроцесорних обчислювальних системах (SMP) має місце практична межа кількості процесорів, які входять в ОС. Ефективна схема з кеш-пам’яттю зменшує трафік шини між процесором і основною пам’яттю, але у міру збільшення числа процесорів трафік шини також зростає. Оскільки шина використовується також для передачі сигналів, що забезпечують когерентність, ситуація з трафіком ще більш ускладнюється. З якогось моменту в плані продуктивності шина перетворюється на вузьке місце. Для систем типу SMP такою межею стає число процесорів в межах від 16 до 64. Наприклад, об’єм SMP-системи Silicon Graphics Power Challenge обмежений 64 процесорами R10000, оскільки при подальшому збільшенні числа процесорів продуктивність падає.

Обмеження на число процесорів в архітектурі SMP служить спонукальним мотивом для розвитку кластерних систем. У останніх же кожен вузол має локальну основну пам’ять, тобто задача «не бачать» глобальної основної пам’яті. По суті, когерентність підтримується не стільки апаратурою, скільки програмним забезпеченням, що не кращим чином позначається на продуктивності. Одним з шляхів створення великомасштабних обчислювальних систем є технологія CC-NUMA. Наприклад, NUMA-система Silicon Graphics Origin підтримує до 1024 процесорів R10000, а Sequent NUMA-Q об’єднує 252 процесори Pentium III.

На рис. 2.16 показана типова організація систем типу CC-NUMA. Є безліч незалежних вузлів, кожен з яких може бути, наприклад, SMP-

системою. Таким чином, вузол містить безліч процесорів, у кожного з яких присутні локальні кеші першого (L1) і другого (L2) рівнів. У вузлі є і основна пам’ять, загальна для всіх процесорів цього вузла, вона розглядається як частина глобальної основної пам’яті системи. У архітектурі CC-NUMA вузол виступає основним будівельним блоком. Наприклад, кожен вузол в системі Silicon Graphics Origin містить два мікропроцесори MIPS R10000, а кожен вузол системи Sequent NUMA-Q включає чотири процесори Pentium III. Вузли об’єднуються за допомогою якої-небудь мережі з’єднань, яка представлена комутованою матрицею, кільцем або має іншу топологію.

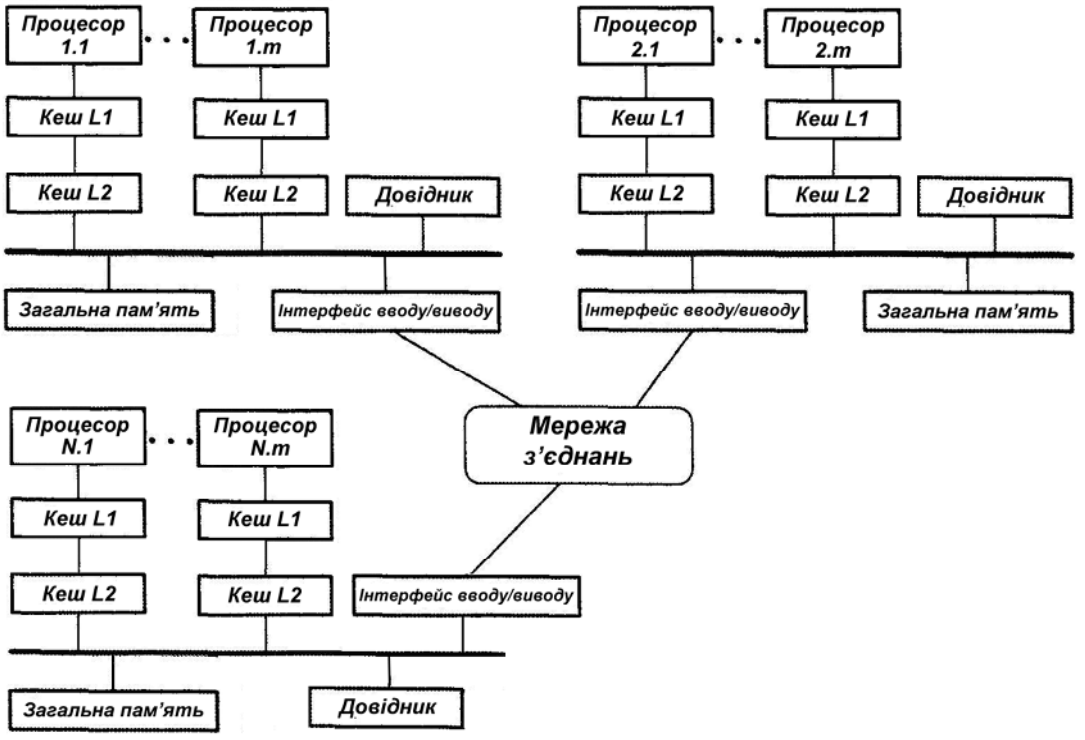


Рис. 2.16. Організація систем типу CC-NUMA

Згідно технології CC-NUMA, кожен вузол в системі володіє власною основною пам’яттю, але з погляду процесорів має місце глобальна адресація пам’яті, де кожен елемент будь-якої локальної основної пам’яті має унікальну системну адресу. Коли процесор ініціює доступ до пам’яті і потрібна комірка відсутнія в його локальній кеш-пам’яті, кеш-пам’ять другого рівня процесора організовує операцію вибірки. Якщо потрібна комірка знаходиться в локальній основній пам’яті, вибірка проводиться з використанням локальної

шини. Якщо ж необхідна комірка зберігається у видаленій секції глобальної пам’яті, то автоматично формується запит, що посилається по мережі з’єднань на потрібну локальну шину і вже по ній до підключеного до даної локальної шини кешу. Всі ці дії виконуються автоматично, прозорі для процесора і його кеш-пам’яті.

У даній конфігурації головна турбота — когерентність кешів. Хоча окремі реалізації і відрізняються в деталях, загальним є те, що кожен вузол містить довідник, де зберігається інформація про місцезнаходження в системі кожної складової глобальної пам’яті, а також про стан кеш-пам’яті.

Щоб проаналізувати, як працює така схема, скористаємося наступним прикладом. Нехай процесор 3 вузла 2 (Р2.3) запрошує комірку з адресою 798, розміщену у вузлі 1. Спостерігатиметься така послідовність дій:

1. Р2.3 видає на шину спостереження вузла 2 запит читання комірки 798.

2. Довідник вузла 2 бачить запит і розпізнає, що потрібна комірка знаходиться у вузлі 1.

3. Довідник вузла 2 посилає запит вузлу 1, який приймається довідником вузла 1.

4. Довідник основної пам’яті 1, діючи як замінник процесора Р2.3, запрошує комірку 798, оскільки ніби він сам є процесором.

5. Основна пам’ять вузла 1 відповідає тим, що надсилає потрібні дані на локальну шину вузла 1.

6. Довідник вузла 1 перехоплює дані з шини.

7. Потрібне значення через мережу з’єднань передається назад в довідник вузла 2.

8. Довідник вузла 2 поміщає отримані дані на локальну шину вузла 2, діючи при цьому як заступник тієї частини пам’яті, де ці дані фактично зберігаються.

9. Дані перехоплюються і передаються в кеш-пам’ять процесора Р2.3 і вже звідти потрапляють в процесор Р2.3.

З опису видно, як дані зчитуються з віддаленої пам’яті за допомогою апаратних механізмів, що роблять транзакції прозорими для процесора. У основі цих механізмів лежить певна форма протоколу когерентності кеш-памяті. Більшість реалізацій відрізняються саме тим, який саме протокол когерентності використовується.

Тема 2.2 Топології обчислювальних систем

2.2.1 Основні поняття ОС

У основі архітектури будь-якої багатопроцесорної обчислювальної системи лежить здатність до обміну даними між компонентами цієї ОС. Комунікаційна система ОС є мережею, *вузли* якої зв’язані трактами передачі даних — *каналами.* В ролі вузлів можуть виступати процесори, модулі пам’яті пристрої вводу/виводу, комутатори або декілька перерахованих елементів, об’єднаних в групу. Організація внутрішніх комунікацій обчислювальної системи називається *топологією.*

Топологію мережі міжз’єднань (ММЗ) визначає безліч вузлів *N*, об’єднаних безліччю каналів *С. Зв’язок* між вузлами зазвичай реалізується по *двоточковій схемі* (point-to-point). Будь-які два вузли, зв’язані каналом зв’язку, називають *суміжними* вузлами або *сусідами.* Кожен канал $c = (x, y)$ є С з’єднує один *вузол-джерело* (source node) *x* з одним *вузлом-одержувачем* (recipient node) *y*, де *x*, *y* є *N.* Вузол-джерело, що виступає початком каналу с, позначатимемо s_c, а вузол-одержувач — другий кінець каналу — r_c. Часто пари вузлів з’єднують два канали — поодинці в кожному напрямі. Канал $c = (x, y)$ характеризується шириною (w_c или w_{xy}) — числом сигнальних ліній; частотою (f_c или f_{xy}) - швидкістю передачі бітів по кожній сигнальній лінії; затримкою (t_c или t_{xy}) — часом пересилки біта з вузла *x* у вузол *y.* Для більшості каналів затримка знаходиться в прямій залежності від фізичної довжини лінії зв’язку (l_c) і швидкості розповсюдження сигналу (v): $l_c = vt_c$. . Смуга пропускання каналу *b c* визначається виразом $b_c = w_c f_c$.

Залежно від того, чи залишається конфігурація взаємозв’язків незмінною, принаймні поки виконується певне завдання, розрізняють мережі із статичною *і* динамічною *топологіями.* У статичних мережах структура взаємозв’язків фіксована. У мережах з динамічною топологією в процесі обчислень конфігурація взаємозв’язків за допомогою програмних засобів може бути оперативно змінена.

Вузол в мережі може бути термінальним, тобто джерелом або приймачем даних, комутатором, що пересилає інформацію з вхідного порту на вихідний, або суміщати обидві ролі. У мережах *з безпосередніми зв’язками* (direct networks) кожен вузол одночасно є як термінальним вузлом, так і комутатором, і повідомлення пересилаються між термінальними вузлами безпосередньо. У мережах *з непрямими зв’язками* (indirect networks) вузол може бути або термінальним, або комутатором, але не одночасно, тому повідомлення передаються опосередковано, за допомогою виділених комутуючих вузлів. (Надалі для простоти викладу дозволимо називати обидва варіанти «прямими» і «непрямими» мережами, також скорочено замість термінального вузла говоритимемо «термінал».) Існують також такі топології, які не можна однозначно зарахувати ні до прямих, ні до непрямих. Будь-яку пряму ММЗ можна зобразити у вигляді непрямої, розділивши кожен вузол на два — термінальний вузол і вузол комутації. Сучасні прямі мережі реалізуються саме таким чином — комутатор відділяється від термінального вузла і поміщається у виділений маршрутизатор. Основна перевага прямих ММЗ в тому, що комутатор може використовувати ресурси термінальної частини свого вузла. Це стає істотним, якщо врахувати, що, як правило, останній включає обчислювальну машину або процесор. Трьома найважливішими атрибутами ММЗ є:

- стратегія синхронізації;
- стратегія комутації;
- стратегія управління.

Дві можливі стратегії синхронізації операцій в мережі — це *синхронна* і *асинхронна*. У синхронних ММЗ всі дії жорстко узгоджені в часі, що забезпечується за рахунок єдиного генератора тактових імпульсів (ГТІ), сигнали якого одночасно транслюються у всі вузли. У асинхронних мережах єдиного генератора немає, а функції синхронізації розподілені по всій системі, причому в різних частинах мережі часто використовуються локальні ГТІ.

Залежно від вибраної стратегії комутації розрізняють *мережі з комутацією з'єднань* і *мережі з комутацією пакетів.* Як у першому, так і в другому варіанті інформація пересилається у вигляді пакету. Пакет *є* групою бітів, для позначення якої застосовують також термін повідомлення.

У мережах з комутацією з'єднань шляхом відповідної установки комутуючих елементів мережі формується тракт від вузла-джерела до вузла-одержувача, що зберігається, поки весь пакет, що доставляється, не досягне пункту призначення. Пересилка повідомлень між певною парою вузлів проводиться завжди поодинці і тому ж маршруту.

Мережі з комутацією пакетів припускають, що повідомлення самостійно знаходить свій шлях до місця призначення. На відміну від мереж з комутацією з'єднань, маршрут від початкового пункту до пункту призначення кожного разу може бути іншим. Пакет послідовно проходить через вузли мережі. Черговий вузол запам'ятовує прийнятий пакет в своєму буфері тимчасового зберігання, аналізує його і робить висновки, що з ним робити далі. Залежно від завантаженості мережі ухвалюється рішення про можливість негайної пересилки пакету до наступного вузла і про подальший маршрут проходження пакету на шляху до мети. Якщо всі можливі тракти для переміщення пакету до чергового вузла зайняті, в буфері вузла формується черга пакетів, яка «розсмоктується» у міру звільнення ліній зв'язку між вузлами (якщо черга також насичується, згідно однієї із стратегій маршрутизації може відбутися так зване «скидання хвоста» (tail drop), відмова від пакетів, що знов поступають).

ММЗ можна також класифікувати по тому, як в них організовано управління. У деяких мережах, особливо з перемиканням з'єднань, прийнято *централізоване управління* (рис. 2.17). Процесори посилають запит на обслуговування в єдиний контролер мережі, який проводить арбітраж запитів з урахуванням заданих пріоритетів і встановлює потрібний маршрут. До даного типу слід віднести мережі з шинною топологією. Процесорні матриці також будуються як мережі з централізованим управлінням, яке здійснюється сигналами від центрального процесора. Приведена схема застосовна і до мереж з комутацією пакетів. Тут тег маршрутизації, що зберігається в заголовку пакету, визначає адреса вузла призначення. Більшість серійних ОС мають саме цей тип управління.

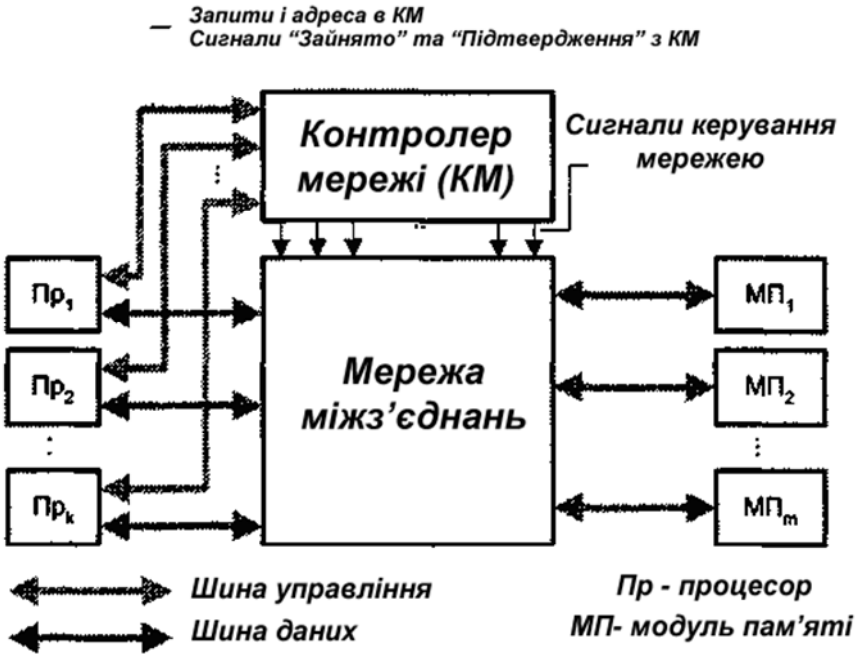


Рис. 2.17 Структура мережі з централізованим управлінням

У схемах з *децентралізованим управлінням* функції управління розподілені по вузлах мережі.

Варіант з централізацією простіше реалізується, але розширення мережі в цьому випадку пов'язане із значними труднощами. Децентралізовані мережі в плані підключення додаткових вузлів значно гнучкіші, проте взаємодія вузлів в таких мережах істотно складніша.

У ряді мереж зв’язок між вузлами забезпечується за допомогою безлічі комутаторів, але існують також мережі з одним комутатором. Наявність великого числа комутаторів веде до збільшення часу передачі повідомлення, але дозволяє використовувати прості перемикальні елементи. Подібні мережі зазвичай будуються як багатоступінчаті.

2.2.2 Методи опису характеристик мережених з’єднань

При описі ММЗ їх зазвичай характеризують за допомогою наступних параметрів:

- розміру мережі (N);
- числа зв’язків (I);
- діаметру (D);
- порядку вузла (d);
- пропускної спроможності (W);
- затримки (T);
- зв’язності (Q);
- ширини бісекції (B);
- смуги бісекції (b).

Розмір мережі (network size) чисельно рівний кількості вузлів, що об’єднуються мережею.

Число зв’язків (number of links) — це сумарна кількість каналів між всіма вузлами мережі. У плані вартості кращою слід визнати ту мережу, яка вимагає меншого числа зв’язків.

Діаметр мережі (network diameter), або *комунікаційна відстань*, визначає мінімальний шлях, по якому проходить повідомлення між двома найбільш віддаленими один від одного вузлами мережі. Шлях в мережі — це впорядкована множина каналів $P = \{c_1, c_2, ..., c_n\}$, по яких дані від вузла-джерела, послідовно переходячи від одного проміжного вузла до іншого, поступають на вузол-одержувач. Для позначення відрізку шляху між парою

суміжних вузлів застосовують термін *перехід*. Мінімальний шлях від вузла *x* до вузла *y* — це шлях з мінімальним числом переходів. Якщо позначити число переходів по мінімальному шляху від вузла *x* до вузла *y* через *H(x, y)*, то діаметр мережі *D* — це найбільше значення *H(x, y)* серед всіх можливих комбінацій *x* і *y*. Так, в ланцюжку з чотирьох вузлів найбільше число переходів буде між крайніми вузлами, і «діаметр» такого ланцюжка рівний трьом. Із зростанням діаметру мережі збільшується загальний час проходження повідомлення, тому розробники ОС прагнуть по можливості обходитися меншим діаметром.

Порядок вузла. Кожен вузол мережі пов'язаний з іншими вузлами множиною каналів $C_{lx} = C_{lx} \cup C_{0x}$, де $C_{lx} = \{C: r_c = x\}$ — множина вхідних каналів, $aC_{0x} = \{c:s_c = x\}$ — множина вихідних каналів. Порядок вузла *x* є сумою числа вхідних $|C_{lx}|$ і вихідних $|C_{0x}|$ каналів вузла, тобто рівний числу вузлів мережі, з якими даний вузол зв'язаний безпосередньо. Наприклад, в мережі, організованій у вигляді матриці, де кожен вузол має канали тільки до найближчих сусідів (зліва, справа, зверху і знизу), порядок вузла рівний чотирьом. Збільшення значення цієї характеристики веде до ускладнення комутаційних пристроїв мережі і, як наслідок, до додаткових затримок в передачі повідомлень. З іншого боку, підвищення порядку вузлів дозволяє реалізувати топології, що мають менший діаметр мережі, і тим самим скоротити час проходження повідомлення. Так, якщо мережа, що складається з 4096 вузлів (2^{12} або 64^2), побудована по матричній схемі, її діаметр складає 126, а для гіперкуба — тільки 12. Розробники ОС зазвичай віддають перевагу таким топологіям, де порядок всіх вузлів однаковий, що дозволяє будувати мережі за модульним принципом.

Пропускна спроможність мережі (network bandwidth) характеризується кількістю інформації, яка може бути передана по мережі в одиницю часу. Зазвичай вимірюється в мегабайтах в секунду або гігабайтах в секунду без урахування витрат на передачу надмірній інформації, наприклад бітів паритету.

Затримка мережі (network latency) — це час, потрібний на проходження повідомлення через мережу. У мережах, де час передачі повідомлень залежить від маршруту, говорять про середню, мінімальну і максимальну затримки мережі.

Зв'язність мережі (network connectivity) можна визначити як мінімальне число вузлів або ліній зв'язку, які повинні вийти з ладу, щоб мережа розпалася на дві непересічні мережі. Отже, зв'язність мережі характеризує стійкість мережі до пошкоджень, тобто її здатність забезпечувати функціонування ОС при відмові компонентів мережі.

Ширина бісекції (bisection width). Спершу визначимо поняття *зрізу мережі* (cut of network). Зріз мережі $C(N_1, N_2)$ — це множина каналів, розрив яких розділяє множина вузлів мережі N на два непересічні набори вузлів N_1 та N_2. Кожен елемент $C(N_1, N_2)$ — це канал, що з'єднує вузол з набору N_1, з вузлом з N_2. Бісекція мережі — це зріз мережі, що розділяє її приблизно навпіл, тобто так, що $|N_2| \le |N_1| \le |N_2| + 1$. Ширину бісекції B характеризують мінімальним числом каналів, що розриваються при всіх можливих бісекціях мережі:

$$B = \min_{bisection} |C(N_1, N_2)|. \qquad (2.1)$$

Ширина бісекції дозволяє оцінити число повідомлень, які можуть бути передані по мережі одночасно, за умови що це не викличе конфліктів через спроби використання одних і тих же вузлів або ліній зв'язку.

Смуга бісекції (bisection bandwidth) — це найменша смуга пропускання по всіх можливих бісекціях мережі. Вона характеризує пропускну спроможність тих ліній зв'язку, які розриваються при бісекції мережі, і дозволяє оцінити якнайгіршу пропускну спроможність мережі при спробі одночасної передачі декількох повідомлень, якщо ці повідомлення повинні проходити з однієї половини мережі в іншу. Смуга бісекції b визначається виразом $b = \min B(N_1, N_2)$. Для мереж з однаковою шириною смуги у всіх b_c каналах справедливе співвідношення: $b = b_c \times B$. Мале

значення смуги бісекції свідчить про можливість конфліктів при одночасній пересилці декількох повідомлень.

2.2.3 Статичні топології

До статичних топологій ММЗ відносять такі, де між двома вузлами можливий тільки один прямий фіксований шлях, тобто статичні топології не припускають наявності в мережі комутуючих пристроїв. Якщо ж такі пристрої є, то використовуються вони тільки перед виконанням конкретного завдання, а в процесі всього часу обчислень топологія ММЗ залишається незмінною.

З можливих критеріїв класифікації статичних мереж найчастіше вибирають їх розмірність. З цих позицій розрізняють:

- одновимірні топології (лінійний масив);
- двовимірні топології (кільце, зірка, дерево, грати, масив систоли); тривимірні топології (повнозв'язна топологія, хордальне кільце); гіперкубічну топологію.

Нижче розглядаються основні види статичних топологій ММЗ без акцентування уваги на якій-небудь їх класифікації, оскільки цей момент для поставленої в посібнику мети неістотний.

Лінійна топологія

У простій *лінійній топології* вузли мережі утворюють одновимірний масив і сполучені в *ланцюжок* (рис. 2.18). Лінійна топологія характеризується наступними параметрами: $D=N-1$; $d=2$; $I=N-1$; $B=1$.

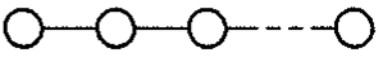


Рис. 2.18. Лінійна топологія

Лінійна топологія не володіє властивістю повної симетричності, оскільки вузли на кінцях ланцюжка мають тільки одну комунікаційну лінію, тобто їх порядок рівний 1, тоді як порядок решти вузлів рівний 2. Час

пересилки повідомлення залежить від відстані між вузлами, а відмова одного з них здатна привести до неможливості пересилки повідомлення. З цієї причини в лінійних ММЗ використовують відмовостійкі вузли, які при відмові ізолюють себе від мережі, дозволяючи повідомленню минути несправний вузол.

Кільцеві топології

Стандартна *кільцева топологія* є лінійним ланцюжком, кінці якого сполучені між собою (рис. 2.19, *а).* Залежно від числа каналів між сусідніми вузлами (один або два) розрізняють *однонаправлені* і *двонаправлені* кільця. Кільцева топологія характеризується наступними параметрами:

$$D = \min\left[\frac{N}{2}\right];\ d = 2;\ I = N;\ B = 2.$$

Кільцева топологія, в порівнянні з лінійною, традиційно була менш популярною, оскільки додавання або видалення вузла вимагає демонтажу мережі.

Один із способів вирішення проблеми великого діаметру кільцевої мережі — додавання ліній зв'язку у вигляді хорд, що з'єднують певні вузли кільця. Подібна топологія носить назву *хордалъної.* Якщо хорди з'єднують вузли з кроком 1 або (*N/2*)-1, діаметр мережі зменшується удвічі. На рис. 2.19, *б* показана *хордальна кільцева мережа з* кроком 3.

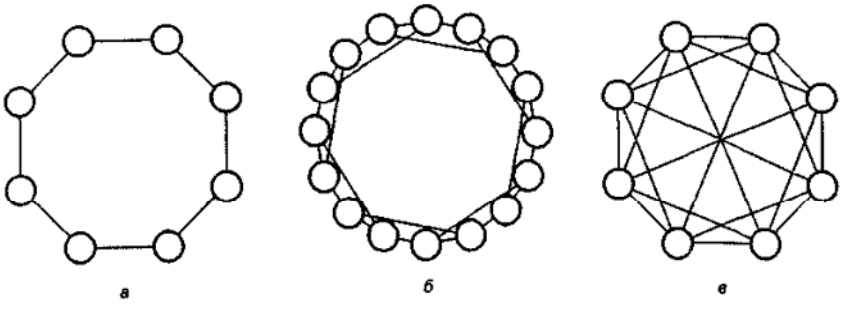


Рис. 2.19. Кільцеві топології: *а* — стандартна; *б* — хордальна; *в* — з циклічним зсувом зв'язків

Як практичні приклади для топології кільця слід назвати мережі Token Ring, розроблені фірмою IBM, а також обчислювальні системи KSR 1 і SCI.

Подальше збільшення порядку вузлів дозволяє добитися ще більшого скорочення тракту передачі повідомлення. Прикладом такої топології може служити показана на рис. 2.3, *в* топологія з *циклічним зсувом зв'язків.* Тут стандартна кільцева топологія з *N* вузлами доповнена з'єднаннями між всіма вузлами *i* і *j*, для яких |*i* - *j*| співпадає з цілим ступенем числа 2. Алгоритми маршрутизації для подібної мережі надзвичайно ефективні, проте порядок вузлів у міру розростання мережі збільшується.

Зіркоподібна топологія

Зіркоподібна мережа об'єднує множину вузлів першого порядку за допомогою спеціалізованого центрального вузла — концентратора (рис. 2.20). Топологія характеризується такими параметрами: D = 2; d = *N*- l; *I* = *N*- l; *B*=l.

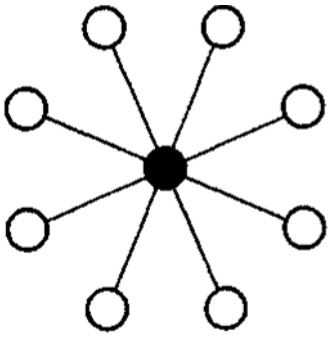


Рис. 2.20 Зіркоподібна топологія

Зіркоподібна організація вузлів і з'єднань рідко використовується для об'єднання процесорів багатопроцесорної ОС, але добре працює, коли потік інформації йде від декількох вторинних вузлів, сполучених з одним первинним вузлом, наприклад при підключенні терміналів. Загальна пропускна спроможність мережі зазвичай обмежується швидкодією концентратора, аналогічно тому, як стримуючим елементом в одношинній топології виступає шина. По продуктивності ці топології також ідентичні. Основна перевага зіркоподібної схеми в тому, що конструктивного виконання вузлів на кінцях мережі може бути дуже простим.

Деревоподібні топології

Ще одним варіантом структури ММЗ є *деревоподібна топологія* (рис. 2.21, а). Мережа будується за схемою так званого строго двійкового дерева, де кожен вузол більш високого рівня зв’язаний з двома вузлами наступного по порядку нижчого рівня. Вузол, що знаходиться на більш високому рівні, прийнято називати *батьківським,* а два підключених до нього розташованих нижче вузла — *дочірними.* У свою чергу, кожен дочірний вузол виступає як батьківський для двох вузлів наступного нижчого рівня. Відзначимо, що кожен вузол пов’язаний тільки з двома дочірними і одним батьківським. Якщо *h* — висота дерева (кількість рівнів в деревовидній мережі), визначається як max[$\log_2$ N], то таку мережу можна охарактеризувати наступними параметрами: *D* = 2(*h-1); d* = 3; I = *N*-1; *B*=1. Так, обчислювальна система з 262 144 вузлів при гратчастій топології (трохи забігаємо вперед) матиме діаметр 512, а у разі строгого бінарного дерева — тільки 36. Топологія двійкового дерева була використана в мультипроцесорній системі DADO з 1023 вузлів, розробленою в Колумбійському університеті.

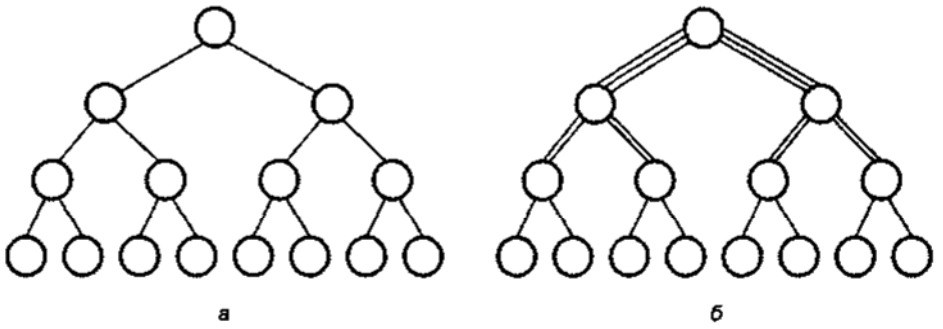


Рис. 2.21. Деревоподібна топологія:
а — стандартне дерево; б — «товсте» дерево

При великих об’ємах пересилок між несуміжними вузлами деревовидна топологія виявляється недостатньо ефективною, оскільки повідомлення повинні проходити через один або декілька проміжних ланок. Очевидно, що на вищих рівнях мережі вірогідність затору через недостатньо високу пропускну здатність ліній зв’язку вище. Цей недолік усувають за допомогою топології «*товстого» дерева* (рис. 2.21, *б*).

Ідея «товстого» дерева полягає в збільшенні пропускної спроможності комунікаційних ліній на прикореневих рівнях мережі. З цією метою на верхніх рівнях мережі батьківські і дочірні вузли зв’язують не одним, а декількома каналами, причому чим вище рівень, тим більше число каналів. На малюнку це відображено у вигляді множинних ліній між вузлами верхніх рівнів. Топологія «товстого» дерева реалізована в обчислювальній системі СМ-5.

Решітчаті топології

Оскільки значна частина науково-технічних задач пов’язана з обробкою масивів, цілком природним представляється прагнення врахувати це і в топології ОС, орієнтованої на подібні задачі. Такі топології відносять до *решітчаттих* (mesh), а їх конфігурація визначається виглядом і розмірністю масиву.

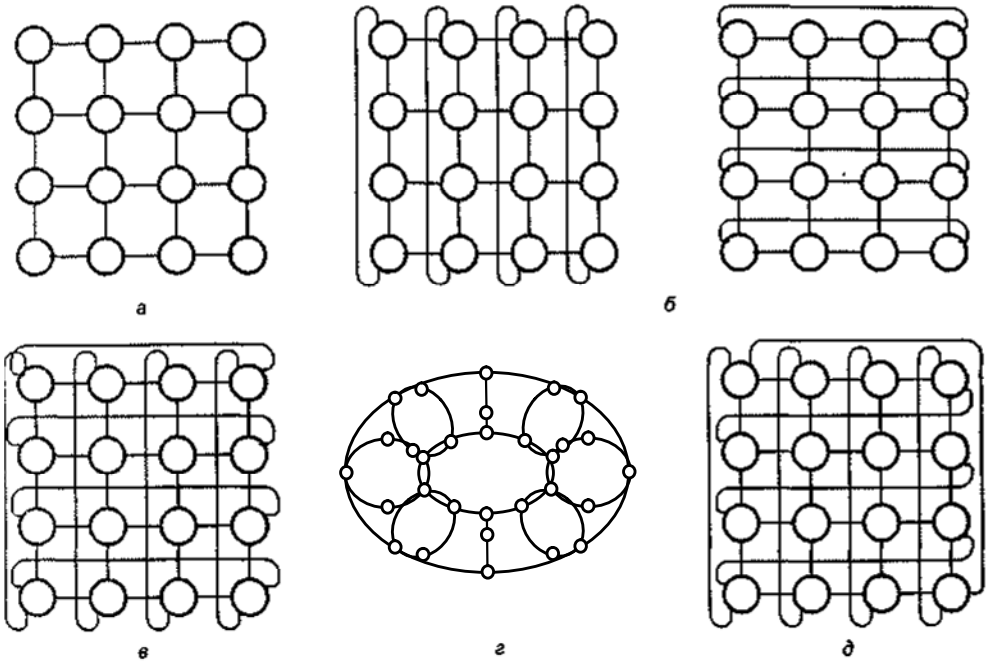


Рис. 2.22. Решітчаті топології: *а* — плоска; *б* — циліндрична; *в-г* — тороїдальна; *д* — вита тороїдальна

Простими прикладами для одновимірних масивів можуть служити лінійка і кільце. Для двовимірних масивів даних найбільш підходить топологія плоскої прямокутної матриці вузлів, кожен з яких сполучений з

найближчим сусідом (рис. 2.22, *а*). Така мережа розмірності *m* x *m* має наступні характеристики: *D = 2*(*m* - 1); d = 4; I = *2N-2m; B = m.*

Якщо провести операцію *згортки* (wraparound) плоскої матриці, з'єднавши інформаційними трактами однойменні вузли лівого і правого стовпців або однойменні вузли верхнього і нижнього рядків плоскої матриці, то з плоскої конструкції отримуємо топологію типу циліндра (рис. 2.22, *б).* У топології циліндра кожний рядок (або стовпець) матриці є кільце. Якщо одночасно провести згортання плоскої матриці в обох напрямах, отримаємо тороїдальну топологію мережі (рис. 2.22, *в*). Двовимірний тор на базі решітки *m* x *m* володіє наступними параметрами:

$$D = 2\min\left[\frac{m}{2}\right];\ d = 4;\ I = 2N;\ B = 2m. \qquad (2.2)$$

Об'ємний вид тороїдальної топології для масиву розмірності 4x8 показаний на рис. 2.22, *г*.

Крім згортки до плоскої решітки може бути застосована операція *скручування* (twisting). Суть цієї операції полягає в тому, що замість кілець всі вузли об'єднуються в розімкнену або замкнуту спіраль, тобто вузли, розташовані з протилежних країв плоскої решітки, з'єднуються з деяким зсувом. Якщо горизонтальні петлі об'єднані у вигляді спіралі, утворюється так звана мережа типу ILLIAC. На рис. 2.22, *д* показана подібна конфігурація ММЗ, відповідна хордальній мережі четвертого порядку, що характеризується наступними параметрами: D = m - 1; d = 4; I =2N; B = *2m.*

Слід згадати і тривимірні мережі. Один з варіантів реалізований в архітектурі суперЕОМ Cray T3D, є тривимірний тор, утворений об'єднанням процесорів в кільця по трьом координатам: *x, y* і *z.*

Прикладами ОС, де реалізовані різні варіанти решітчатих топологій, можуть служити: ILLIAC IV, MPP, DAP, CM-2, Paragon та ін.

Повнозв’язна топологія

У *повнозв’язній топології* (рис. 2.23), відомій також під назвою топології «*максимального групування*» кожен вузол безпосередньо сполучений з рештою всіх вузлів мережі. Мережа, що складається з N вузлів, має наступні параметри:

$$D = 1;\ d = N - 1;\ I = \frac{N(N-1)}{2};\ B = \frac{N^2}{4}. \qquad (2.3)$$

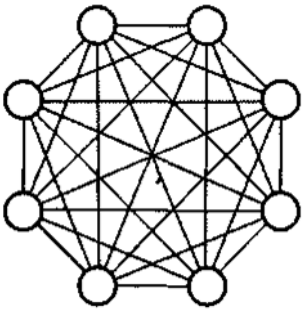


Рис. 2.23 Повнозв’язна топологія

Якщо розмір мережі великий, топологія стає дорогою і такою, що важко реалізовується. Більш того, повнозв’язна топологія не дає істотного поліпшення продуктивності, оскільки кожна операція пересилки вимагає, щоб вузол проаналізував стан всіх своїх *N* - 1 входів. Для прискорення цієї операції необхідно, щоб всі входи аналізувалися паралельно, що, у свою чергу, ускладнює конструкцію вузлів.

Топологія гіперкуба

При об’єднанні паралельних процесорів часто використовується топологія гіперкуба, показана на рис. 2.24. Лінія, що з’єднує два вузли (рис. 2.24, *а)*, визначає одновимірний гіперкуб. Квадрат, утворений чотирма вузлами (рис. 2.24, *б*), — двовимірний гіперкуб, а куб з 8 вузлів (рис. 2.24, *в*) — тривимірний гіперкуб і так далі. З цього ряду виходить алгоритм отримання *m*-мірного гіперкуба: починаємо з *(m-1*) -мірного гіперкуба, робимо його ідентичну копію, а потім додаємо зв’язки між кожним вузлом

початкового гіперкуба і однойменним вузлом копії. Гіперкуб розмірності m ($N= 2^m$) має наступні характеристики:

$$D = m;\ d = m;\ I = \frac{mN}{2};\ B = m. \qquad (2.4)$$

Збільшення розмірності гіперкуба на 1 веде до подвоєння числа його вузлів, збільшення порядку вузлів і діаметру мережі на одиницю.

Обмін повідомленнями в гіперкубі базується на двійковому представленні номерів вузлів. Нумерація вузлів проводиться так, що для будь-якої пари суміжних вузлів двійкове представлення номерів цих вузлів відрізняється тільки в одній позиції. Відповідно до сказаного, вузли 0010 і 0110 — сусіди, а вузли 0110 і 0101 – ні. Простий спосіб нумерації вузлів при створенні *m*-мірного гіперкуба з двох *(m* - 1) -мірних показаний на рис. 2.8, *д*. При копіюванні *(m* - 1) -мірного гіперкуба і з'єднанні його з початковим *(m* - 1) -мірним гіперкубом необхідно, щоб вузли, що з'єднуються, мали однакові номери. Далі до номерів вузлів початкового гіперкуба зліва додається біт, рівний 0, а до номерів вузлів копії — одиничний біт.

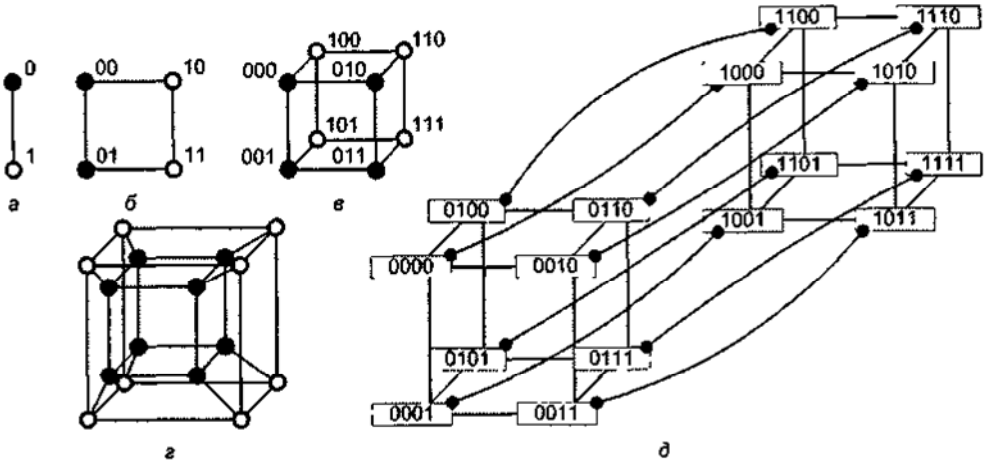


Рис. 2.24. Топологія гіперкуба:
а — одновимірна; б — двохвимірна; в — тривимірна; г — чотиривимірна;
д — схема формування чотиривимірного гіперкуба з двох тривимірних

Номери вузлів є основою маршрутизації повідомлень в гіперкубі. Такі номери в m-мірному гіперкубі складаються з *m* бітів, а пересилка

повідомлення з вузла *A* у вузол *B* виконується за *m* кроків. На кожному кроці вузол може або зберегти повідомлення і не пересилати його далі до наступного кроку, або відправити його далі по одній з ліній. На кроці *i* вузол, що зберігає повідомлення, порівнює *i-й* біт свого власного номера з *i-м* бітом номера вузла призначення. Якщо вони співпадають, передавання повідомлення припиняється, якщо немає — повідомлення передається уздовж лінії *i*-го виміру. Лінією *i*-го виміру вважається та, яка була додана на етапі побудови *i*-мірного гіперкуба з двох (*i* - 1) - мірних.

Створення гіперкуба при великому числі процесорів вимагає збільшення порядку вузлів, що зв'язане з великими технічними проблемами. Компромісне рішення, що збільшує діаметр мережі при збереженні базової структури, є куб з циклічно з'єднаних вузлів (рис. 2.25). Тут порядок вузла рівний трьом при будь-якому розмірі мережі.

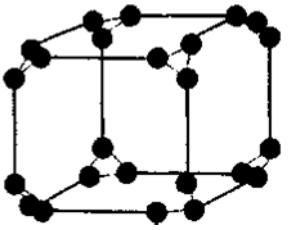


Рис. 2.25 Куб з циклічно з'єднаних вузлів третього порядку

Топологія *k*-мірного *n*-куба

Назва топології означає, що в ній реалізується куб, що має *n* вимірів, причому кожен вимір містить *k* вузлів *($N = k^n$)*. Кожному вузлу призначений *n*-розрядний номер в системі числення з основою *k*, і він пов'язаний з вузлом, номер якого відрізняється тільки в одній позиції і лише на одиницю. *k*-мірний *n*-куб може бути побудований шляхом об'єднання *k* екземплярів *k*-мірних (*n* - 1) - кубів в кільце.

Раніше розглянуті топології є варіантами топології *k*-мірного *n*-куба *:*

k-мірний 1-куб — кільце;

k-мірний 2-куб — двовимірний тор;

k-мірний 3-куб — тривимірний тор;

4-мірний 2-куб — плоска решітка 4x4;

2-мірний *n*-куб — гіперкуб.

Доведено, що ефективність топології, а також її масштабованість поліпшуються із зростанням значення k і зменшенням кількості вимірів n.

Дані по розглянутих статичних топологіях зведені в табл. 2.2.

Таблиця 2.2. Характеристики мереж із статичною топологією

Топологія	Діаметр	Порядок вузла	Число зв'язків	Ширина бісекції	Симетрич-ність	Розмір мережі
Повнозв'язна	1	N-1	$\frac{N(N-1)}{2}$	$\frac{N^2}{4}$	Так	N вузлів
Зірка	2	1	$N-1$	1	Немає	N вузлів
Двійкове дерево	$2(h-1)$	3	$N-1$	1	Немає	Висота дерева $h = \log_2 N$
Лінійний масив	$N-1$	2	N - 1	1	Немає	N вузлів
Кільце	$\frac{N}{2}$	2	N	2	Так	N вузлів
Двовимірна решітка	2(m-1)	4	$2N-2m$	$\sqrt{N}$	Немає	Грати m x m, де $m =$
Двовимірний тор	2[m/2]	4	$2N$	$2m$	Так	Тор m x m, де $m = \sqrt{N}$
Гіперкуб	n	n	$\frac{nN}{2}$	$\frac{N}{2}$	Так	N вузлів; $n = \log_2 N$
k-мірний n-куб	n[k/2]	$2n$	nN	$2k^{n-1}$	Так	$N=k^n$ вузлів

2.2.4 Динамічні топології

У динамічній топології мережі з'єднання вузлів забезпечується електронними ключами, варіюючи установки яких можна міняти топологію мережі. На відміну від раніше розглянутих топологій, де роль вузлів грають самі об'єкти інформаційного обміну, у вузлах динамічних мереж розташовуються комутуючі елементи. Пристрої, що обмінюються повідомленнями (термінали), підключаються до входів і виходів цієї мережі.

В ролі терміналів можуть виступати процесори або процесори і модулі пам’яті.

Якщо входи і виходи мережі комутуючих елементів розділені, мережу називають двосторонньою (two-sided). При суміщених входах і виходах мережа є односторонньою (one-sided).

Зазвичай ключі в динамічних ММЗ групуються в так звані *ступені комутації.* Залежно від того, скільки ступенів комутації містить мережа, вона може бути одноступінчатою або багатоступінчатою. Наявність більш ніж одного ступеню комутації дозволяє забезпечити множинність шляхів між будь-якими парами входів і виходів.

Мінімальною вимогою до мережі з комутацією є підтримка з’єднання будь-якого входу з будь-яким виходом. Для цього в мережі з n входами і n виходами система ключів зобов’язана надати $n!$ варіантів комутації входів і виходів (перестановок — permutations). Проблема ускладнюється, коли мережа повинна забезпечувати одночасну передачу даних між багатьма парами термінальних вузлів (multicast), причому так, щоб не виникали конфлікти (блокування) через передавання даних через одні і ті ж комутуючі елементи в один і той же час. Подібні топології повинні підтримувати n^n перестановок. З цих позицій всі топології ММЗ з комутацією розділяються на три типи: неблокуючі, неблокуючі з реконфігурацією і блокуючі.

У неблокуючих мережах забезпечується з’єднання між будь-якими парами вхідних і вихідних терміналів без перенастроювання комутуючих елементів мережі. В рамках цієї групи розрізняють мережі строго неблокуючі (strictly non-blocking) і неблокуючі в широкому сенсі (wide sense non-blocking). У строго неблокуючих мережах виникнення блокувань принципово неможливе через застосовану топологію. До таких відносяться матрична мережа і мережа Клоша. Неблокуючими в широкому сенсі називають топології, в яких конфлікти при будь-яких з’єднаннях не виникають тільки при дотриманні певного алгоритму маршрутизації.

У неблокуючих мережах з реконфігурацією також можлива реалізація з’єднання між довільними вхідними і вихідними терміналами, але для цього необхідно змінити настройку комутаторів мережі і маршрут зв’язку між з’єднаними терміналами. Прикладами таких мереж служать мережі Бенеша, Бетчера, «Мемфіс» та ін.

У блокуючих мережах, якщо яке-небудь з’єднання вже встановлене, це може стати причиною неможливості встановлення інших з’єднань. До блокуючих відносяться мережі «Баньян», «Омега» та ін.

Шинна топологія

Мережі з шинною архітектурою — найбільш простий і дешевий вид динамічних мереж. При одношинній топології, показаній на рис. 2.26, *а*, всі вузли мають порядок 1 (d=1) і підключені до однієї спільно використовуваної шини. У кожен момент часу обмін повідомленнями може вести тільки одна пара вузлів, тобто на період передачі повідомлення шину можна розглядати як мережу, що складається з двох вузлів, через що її діаметр завжди рівний 1 (D = 1). Також одиниці рівна і ширина бісекції (B), оскільки топологія допускає одночасну передачу тільки одного повідомлення. Одношинна конфігурація може бути корисною, коли число вузлів невелике, тобто коли трафік шини малий в порівнянні з її пропускною спроможністю. Одношинну архітектуру часто використовують для об’єднання декількох вузлів в групу (кластер), після чого з таких кластерів утворюють мережу на базі інших видів топології.

Багатошинна топологія допускає наявність n незалежних шин і підключення вузлів до кожної з цих шин (рис. 2.26, *б*), що дозволяє вести одночасну пересилку повідомлень між n парами вузлів. Така топологія цілком придатна для високопродуктивних ОС. Діаметр мережі як і раніше рівний 1, тоді як пропускна спроможність зростає пропорційно числу шин. В порівнянні з одношинною архітектурою управління мережею з декількома шинами складніше через необхідність запобігання конфліктам, що

виникають, коли в парах вузлів, що обмінюються по різних шинах, присутній загальний вузол. Крім того, із збільшенням порядку вузлів стає складніша їх технічна реалізація.

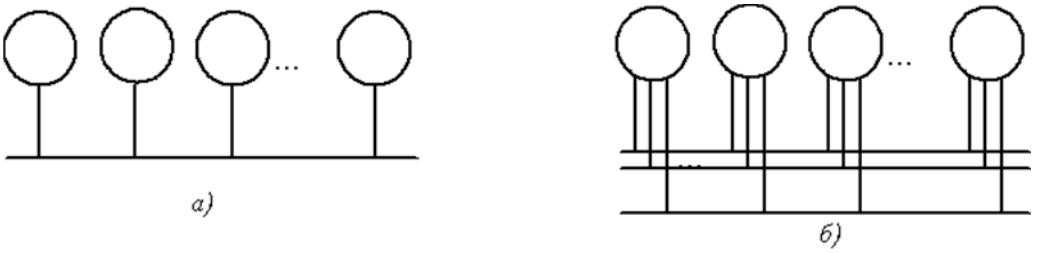


Рис. 2.26 Шинна топологія: а - з однією шиною; б - з багатьма шинами

Топологія перехресної комутації («кросбар»)

Топологія перехресної комутації мультипроцесорної системи (crossbar switch systein) на основі матричного (координатного) комутатора є класичним прикладом одноступінчатої динамічної мережі. Не зовсім офіційний термін «кросбар», який застосовуватиметься надалі для позначення даної топології, бере свій початок з механічних координатних (крокових) шукачів, що використалися на зорі телефонії. Кросбар *n* x *m* (рис. 2.27) – це комутатор, здатний з'єднати *n* вхідних і *m* вихідних термінальних вузлів з рівнем паралелізму, рівним min*(n, m).* Головна перевага даної топології полягає тому, що мережа виходить неблокуючою і забезпечує меншу затримку в передачі повідомлень в порівнянні з іншими топологіями, оскільки будь-який шлях містить тільки один ключ. Однак, через те, що кількість ключів в мережі рівна *n* x *m,* використання кросбара у великих мережах стає непрактичним, хоча це достатньо хороший вибір для малих мереж. Нижче буде показано, що для великих неблокуючих мереж можна запропонувати інші топології, що вимагають істотно меншої кількості ключів.

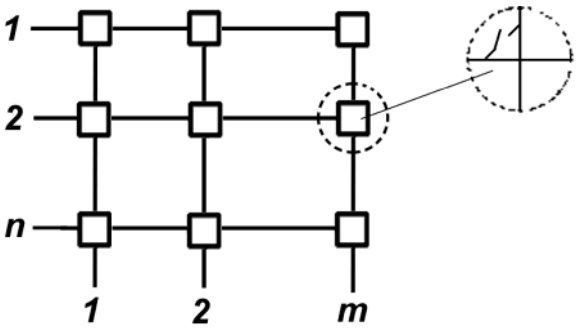


Рис. 2.27 Матричний комутатор *n* x *m*

Якщо *n* = *m*, то такий комутатор називається «повний кросбар». «Повний кросбар» на *n* входів і *n* виходів містить n^2 ключів. Діаметр мережі рівний 1, ширина бісекції — *n/2*. Цей варіант часто використовують в мережах з деревовидною топологією для об'єднання вузлів нижнього рівня, роль яких грають невеликі групи (кластери) процесорів і модулів пам'яті.

Сучасні комерційно доступні матричні комутатори здатні з'єднувати до 256 пристроїв. Топологія використовується для організації з'єднань в деяких обчислювальних системах, що серійно випускаються, наприклад в Fujitsu VPP500 224 x 224.

Комутуючі елементи мереж з динамічною топологією

Оскільки топології, які будуть надалі розглядатися відносяться до багатоступінчатих, то спочатку необхідно визначити типи комутуючих елементів, що використовуються в ступенях комутації таких мереж. За цією ознакою розрізняють:

- мережі на основі перехресної комутації;
- мережі на основі базового комутуючого елементу.

У мережах, що відносяться до першої групи, як базовий комутуючий елемент використовується кросбар *n* x *m*. Для другої категорії роль комутуючого елементу грає «повний кросбар» 2x2. Потенційно такий комутатор управляється чотирирозрядним двійковим кодом і забезпечує 16 варіантів комутації, з яких корисними можна вважати 12. На практиці ж зазвичай задіюють тільки чотири можливі стани кросбара 2x2, які

визначаються дворозрядним керуючим кодом (рис. 2.28). Подібні кросбари називають базовим комутуючим елементом (БКЕ) або *β*-елементом. Перші два стани БКЕ є основними: у них вхідна інформація може транслюватися на виходи прямо або навхрест. Два наступні стани призначено для широкомовного режиму, коли повідомлення від одного вузла одночасно транслюється на всі підключені до нього інші вузли. Широкомовний режим використовується рідко. Сигнали на перемикання БКЕ в певний стан можуть формуватися пристроєм управління мережею. У складнішому варіанті БКЕ ці сигнали формуються усередині самого *β*-елемента, виходячи з адрес пунктів призначення, що містяться у вхідних повідомленнях.

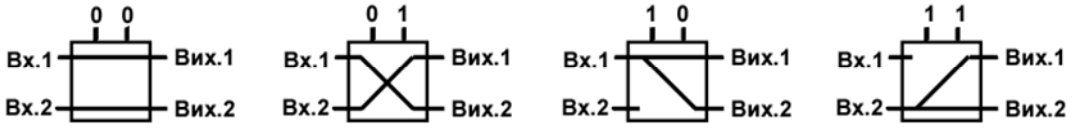


Рис. 2.28. Стани *β*-елемента

Структура *β*-елемента показана на рис. 2.29.

Вибір на користь того або іншого варіанту комутації вхідних повідомлень (пакетів) здійснюється схемою логіки прийняття рішення *β*-елемента. Конкретний вид комутації реалізується здвоєним мультиплексором, керованим з виходу регістра, де зберігається результат роботи схеми логіки прийняття рішення. Елементи затримки забезпечують синхронізацію процесів прийняття рішення і пересилки пакетів з входів на виходи.

Складність β-елемента визначається від логікою прийняття рішення. У ряді архітектур БКЕ їх стан визначається тільки бітом активності пакету. У інших архітектурах використовуються адреси джерела і одержувача даних, що зберігаються в заголовку пакету, а це може вимагати підтримки в БКЕ спеціальних таблиць. Проте у всіх своїх варіантах β-елементи достатньо прості, що дозволяє реалізувати їх на базі інтегральних мікросхем.

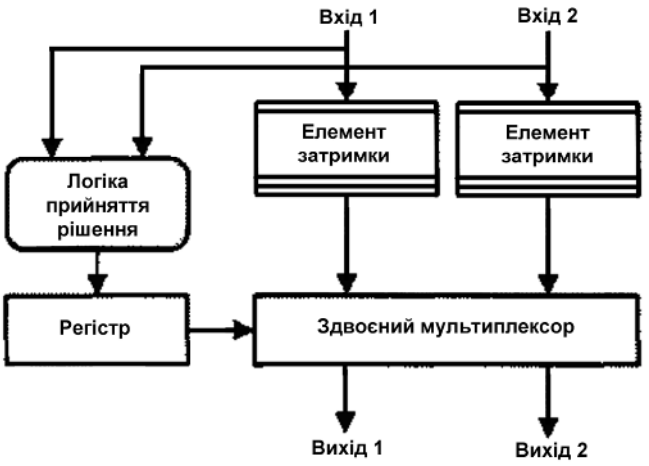


Рис. 2.29 Структура β -елемента

2.2.5 Функції маршрутизації даних в динамічних топологіях

Кардинальним питанням при виборі топології ММЗ є спосіб маршрутизації даних, тобто правило вибору чергового вузла, якому пересилається повідомлення. Основою маршрутизації служать адреси вузлів. Кожному вузлу в мережі присвоюється унікальна адреса. Виходячи з цих адрес, а точніше, їх двійкових представлень, проводиться з'єднання вузлів в статичних топологіях або їх комутація в динамічних топологіях. По суті, прийнята система відповідності між двійковими кодами адрес суміжних вузлів — функція маршрутизації даних — і визначає мережеву топологію. Останню можна описати як набір функцій маршрутизації, що задає порядок вибору проміжних вузлів на шляху від вузла-джерела до вузла-одержувача. У деяких топологіях використовується єдина для всієї ММЗ функція маршрутизації, в інших — багатоступінчатих — при переході від одного ступеня до іншої може застосовуватися інша функція маршрутизації.

Функція маршрутизації даних задає правило обчислення можливої адреси однієї з суміжних вузлів за адресою другого вузла. Зводиться це до опису алгоритму маніпуляції бітами адреси-джерела для визначення адреси-одержувача. Нижче приводиться формальний опис основних функцій маршрутизації даних, які використовують у відомих топологіях ММЗ, без аналізу їх переваг і недоліків. Для всіх функцій передбачається, що

розмірність мережі рівна N, а розрядність адреси — m, де $m = log_2 N$. Біти адреси позначені як b_i.

Перестановка

Функція перестановки (exchange) відповідає наступному співвідношенню:

$$E_k(b_m, ..., b_k, ..., b_1) = (b_m, ..., \overline{b_k}, ..., b_1), \ 1 \le k \le m. \qquad (2.5)$$

Працюючим прикладом для даної функції маршрутизації може служити топологія гіперкуба (рис. 2.30).

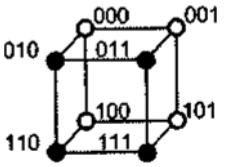


Рис. 2.30 Приклад топології з функцією перестановки

Тасування

Функція тасування (shuffle) може бути реалізована в одному з чотирьох варіантів: ідеальне тасування (perfect shuffle), відсутність тасування (unshuffle), субтасування по i-му біту (i^{th} subshuffle) і супертасування по i-му біту (i^{th} supershuffle). Нижче приведені формальні описи кожного з перерахованих варіантів, а на рис. 2.31 — приклади відповідних їм топологій.

Ідеальне тасування:

$$S(b_m, b_{m-1}, ..., b_1) = (b_{m-1}, b_{m-2}, ..., b_1, b_m). \qquad (2.6)$$

З приведеної формули видно, що два вузли з адресами i і j мають між собою безпосередній зв'язок за умови, що двійковий код j може бути отриманий з двійкової коду i циклічним зсувом вліво. Ідеальне тасування — найбільш поширений серед даних варіантів функції тасування.

Відсутність тасування:

$$U(b_m, b_{m-1}, ..., b_1) = (b_1, b_m, ..., b_2). \quad (2.7)$$

Субтасування по *i*-му біту:

$$S_i(b_m, b_{m-1}, ..., b_i, ..., b_1) = (b_m, ..., b_{i+1}, b_{i-1}, ..., b_1, b_i). \quad (2.8)$$

Супертасування по *i*-му біту:

$$S^i(b_m, b_{m-1}, ..., b_i, ..., b_1) = (b_i, b_{m-1}, ..., b_{i+1}, b_m, b_{i-1}, ..., b_1). \quad (2.9)$$

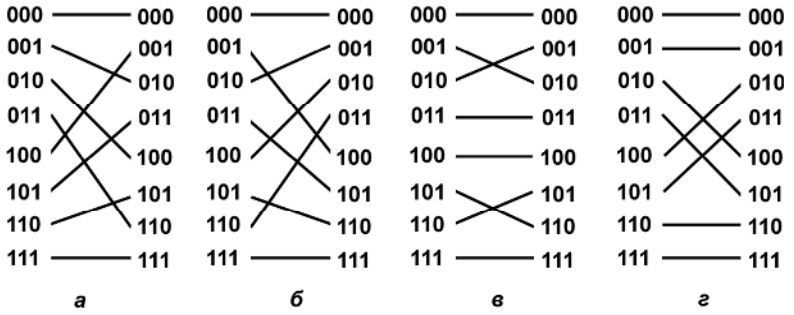


Рис. 2.31. Приклади топологій з тасуванням для m = 3:
***а* — ідеальне тасування; *б* — відсутність тасування;**
***в* — субтасування по другому біту; *г* — супертасування по другому біту**

Батерфляй

Функція «батерфляй» (butterfly) — «метелик» була розроблена в кінці 60-х років Рабінером і Гоулдом. Свою назву вона отримала через те, що побудована відповідно до неї мережа по конфігурації нагадує крила метелика (рис. 2.32). Математично функція може бути записана у вигляді:

$$B(b_m, b_{m-1}, ..., b_1) = (b_1, b_{m-1}, ..., b_2, b_m). \quad (2.9)$$

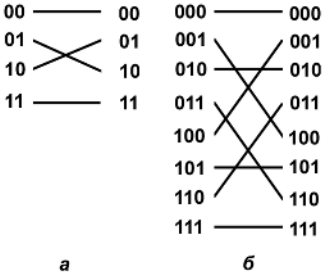


Рис. 2.32 Приклади топології «батерфляй» для:
а - для *m* = 2; *б* - для *m* = 3.

Хоча функція «батерфляй» використовується в основному при об’єднанні ступенів в мережах з динамічною багатоступінчатою топологією, відомі також і «чисті» «батерфляй»-мережі.

Реверсування бітів

Як випливає з назви, функція зводиться до перестановки бітів адреси в зворотному порядку:

$$R(b_m, b_{m-1}, ..., b_1) = (b_1, b_2, ..., b_m). \quad (2.9)$$

Відповідна топологія для m = 3 показана на рис. 2.33. Хоча для значення m = 3 топологія реверсування бітів співпадає з топологією «батерфляй», при великих значеннях m відмінності стають очевидними.

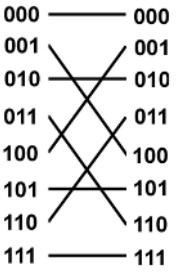


Рис. 2.33 Приклад топології на основі формули реверсування бітів (m = 3)

Зсув

Функція маршрутизації по алгоритму зсуву має вигляд:

$$SX(x) = (x+1) \bmod N \quad (N = 2^m) \quad (2.9)$$

При m = 3 дана функція відповідає топології кільця (рис. 2.34).

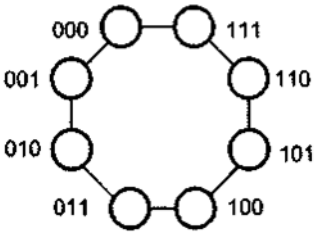


Рис. 2.34 Приклад топології на основі функції зсуву (m = 3)

Мережа ILLIAC IV

Комбінуючи декілька варіантів функції зсуву, можна утворити складніші функції маршрутизації. Найбільш відомою з таких «складних» функцій є мережа ILLIAC IV, вперше реалізована в топології обчислювальної системи ILLIAC IV:

$$\begin{gathered} R_{+1} = (i+1) \bmod N; \\ R_{-1} = (i-1) \bmod N; \\ R_{+r} = (i+r) \bmod N \quad (0 \le i \le N-1); \\ R_{-r} = (i-r) \bmod N \quad (r = \sqrt{N}). \end{gathered} \tag{2.10}$$

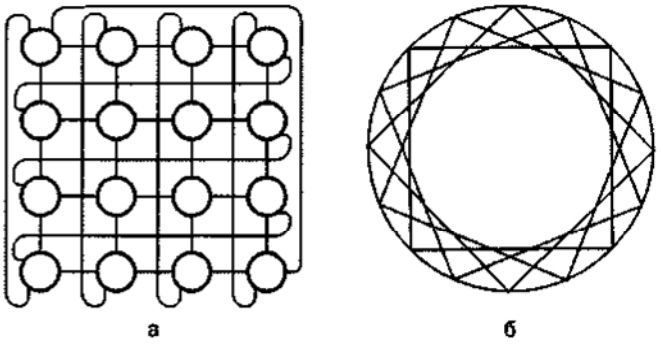


Рис. 2.35 Топологія мережі ILLIAC IV:
***а* — у вигляді решітки; *б* — у вигляді хордального кільця**

Приведені співвідношення для *N=4* відповідають двом варіантам топології, показаним на рис. 2.35.

Перший варіант є фігурою, побудованою на базі плоскої решітки, де вузли в кожному стовпці замкнуті в кільце, а вузли в послідовних рядах сполучені в замкнуту спіраль. Другий варіант відповідає хордальному кільцю з кроком хорди, рівним 4.

Циклічний зсув

Функція циклічного зсуву (barrel shift) описується виразами

$$\begin{gathered} B_{+1}(j) = (j + 2^i) \bmod N, \\ B_{-1}(j) = (j - 2^i) \bmod N, \text{де } 0 \le j \le N-1,\ 0 \le i \le \log_2 N - 1. \end{gathered} \tag{2.11}$$

Топологію мережі на базі даної функції маршрутизації даних ілюструє рис. 2.36.

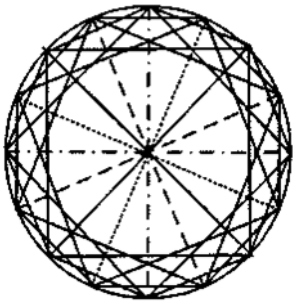


Рис. 2.36 Приклад топології на основі функції циклічного зсуву ±2i

Тема 2.3. Відмовостійкі ПРКС

2.3.1 Основні поняття відмовостійкості

Системи із статичною надлишковістю для виявлення відмов і збоїв типу Tandem мають обмежену область застосування. Популярніші кластери і MPP системи, що будуються по концепції відкритих систем і мають доступну всім процесорам файлову систему. Вони володіють наступними характеристиками:

• виглядають для користувача як єдина система (управляються і адмініструються як єдина система; мають єдину файлову систему);

• мають високу готовність;

• масштабуються;

• гнучко перебудовуються при зміні задач.

Як правило, всі комп'ютери в паралельній системі виконують корисну роботу, а не знаходяться в гарячому резерві. У разі збою додатку, він може бути перезапущений на іншому вузлі системи.

ОС обов'язково має загальнодоступні ресурси, до числа яких зазвичай входять дискові накопичувачі, що дозволяють перезапускати додатки на різних комп'ютерах, зв'язаних комунікаційним середовищем. Файлова системи використовує журналювання, незалежний буфер дисків і системи резервного копіювання.

Використовуване комунікаційне середовище визначається додатком. Так, якщо потрібна лише перевірка працездатності машин кластера і відстежування його конфігурації, то середовище може мати низьку пропускну спроможність на рівні 10 Мбіт/с (Ethernet). Якщо необхідне використання в різних вузлах даних спільних дискових масивів, то вимоги до пропускної спроможності комунікаційного середовища ОС істотно зростають.

Як шина для підключення зовнішніх пристроїв ОС застосовується SCSI інтерфейс, який забезпечує 80 Мбайт/с, а Ultra SCSI забезпечує 160 Мбайт/с.

В останні роки дедалі більшої популярності набуває високошвидкісний інтерфейс Serial Attached SCSI (SAS), який підтримує швидкість обміну даними до 3 Гбіт/с, і очікується збільшення швидкості передавання до 10 Гбіт/с.

Різні моделі відмовостійких систем

Гарячий резерв. Структура системи показана на рис. 2.37.

Один сервер виконує додатки, інший знаходиться в гарячому резерві. Дискові масиви підключені до обох серверів. Спеціальні програми, використовуючи виділену мережу, стежать за станом один одного. При виявленні збоїв або відмови після закінчення таймаутного інтервалу додатки перезапускаються на тому ж сервері, якщо він визнаний справним, або на резервному сервері з використанням стану, збереженого в дисковому масиві. Резервний сервер може бути менш потужнім і виконувати тільки критичні програми.

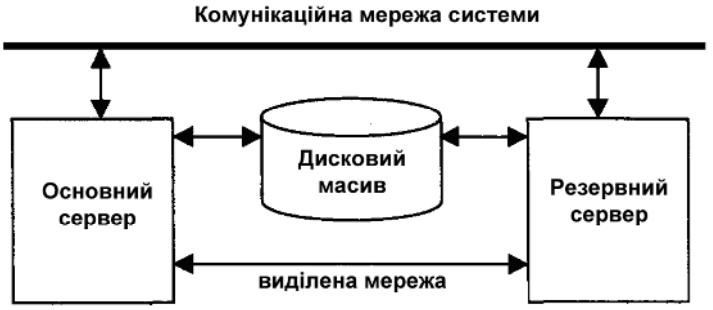


Рис. 2.37 Гарячий резерв

Реплікація. Структура системи з копіюванням (реплікацією) даних показана на рис. 2.38.

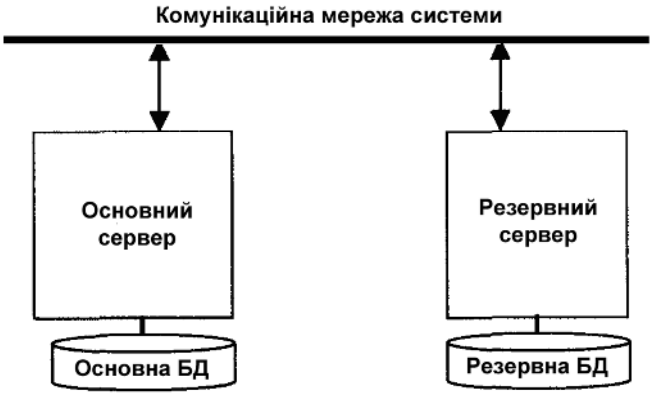


Рис. 2.38 Відмовостійка система з реплікацією

У нормальному режимі основний сервер виконує додатки і копіює стан основної бази даних в резервну базу даних після закінчення кожної транзакції, або після закінчення заданого інтервалу часу. У разі відмови, наприклад, після закінчення тайаутного інтервалу, після чергової реплікації даних програмні додатки запускаються на резервному сервері.

Паралельний сервер бази даних. Структура показана на рис. 2.39.

Використовуються СУБД, що реалізовують розподілену і паралельну обробку, наприклад Teradata, Oracle або Informix, які функціонують як єдина система на всіх ВМ кластера. При цьому монітор дискової системи управляє доступом до дисків і контролює працездатність устаткування дискових масивів. Виділена мережа використовується для взаємного стеження комп'ютерів за роботою один одного. Дзеркалювання дисків, тобто підтримка двох абсолютних копій дисків, дозволяє у разі відмови одного диска відразу використовувати інший для продовження функціонування, не витрачаючи час на перезапуск застосувань.

Відмовостійкі MPP системи. Основна відмінність від попередніх систем полягає в повній незалежності всіх ресурсів, так званому sharing nothing. Це дозволяє парирувати майже всі випадки відмов. Програми працюють з сервером програм, який формує транзакції до різних баз даних. Сервер програм побудований як відмовостійка система спільно працюючих і взаємодіючих процесів.

Кожен комп'ютер може бути зупинений для профілактики, а також

заміни устаткування і програмного забезпечення без порушення працездатності системи.

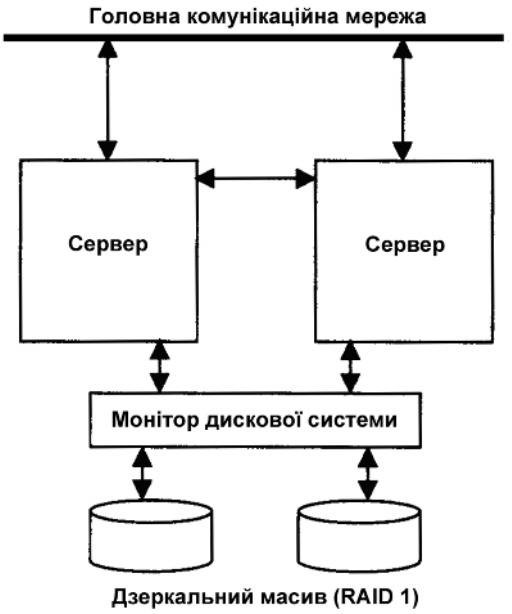


Рис.2.39 Відмовостійка реалізація паралельного сервера БД

2.3.2 Забезпечення відмовостійкості дискової пам’яті

Однією із найчастіших проблем, які виникають при роботі комп’ютерних систем є відмова або збій роботи дискових накопичувачів (HDD). Це може статися як внаслідок випадкових збоїв у роботі контролера диску, так і при механічному пошкодженні поверхні диску. Тому для високонадійних систем дуже часто звертається увага на вживання заходів, які дозволяють збільшити відмовостійкість саме дискової пам’яті. З цією метою найчастіше використовуються технології RAID.

У перекладі з англійської «RAID» (Redundant Arrays of Inexpensive Disks) означає «надлишковий масив незалежних дисків». Вперше термін RAID з’явився в 1987 році, коли дослідниками з Каліфорнійського Університету в Берклі вдалося створити масив, що складався з декількох жорстких дисків.

Первинне призначення RAID – створення на базі декількох вінчестерів диска великого об’єму із збільшеною швидкістю доступу. Але потім до двох основних цілей додалася третя – збереження даних у разі відмови частини устаткування. Саме ці три задачі зробили RAID-масиви такими затребуваними бізнесом і військовими. Втім, за об’єм, швидкість і надійність довелося платити підвищенням вартості і складності систем зберігання даних.

З часом устаткування для побудови RAID масивів стало доступнішим, особливо з появою дешевих рішень для IDE/ATA і SATA дисків. Тепер уже не тільки фахівці з високонадійних систем, але і звичайні користувачі зіткнулися з хитрощами побудови дискових масивів.

Виявляється, не так просто знайти оптимальне рішення одночасно по надійності, ємкості і ціні. Треба бути готовим до того, що доведеться купити не один, а декілька жорстких дисків, і ємкість як мінімум одного з них не використовуватиметься. Якщо мова йде про побудову більш-менш серйозної системи, буде потрібно окремий (краще спеціальний) корпус з окремим (а то і двома) блоком живлення, плата контроллера і відповідне програмне забезпечення.

У основі теорії RAID лежать п’ять основних принципів – масив (Array), дзеркалювання (Mirroring), дуплекс (Duplexing), чергування (Striping) і парність (Parity).

Масивом називають декілька накопичувачів, які централізовано настроюються, форматуються і управляються. Логічний масив – це вже вищий рівень представлення, на якому не враховуються фізичні характеристики системи. Відповідно, логічні диски можуть по кількості і об’єму не співпадати з фізичними. Але краще все-таки дотримувати відповідність: фізичний диск – логічний диск. Нарешті, для операційної системи взагалі весь масив є одним великим диском.

Дзеркалювання – технологія, що дозволяє підвищити надійність системи. У RAID масиві із дзеркалюванням всі дані одночасно пишуться не

на один, а на два жорсткі диски. Тобто створюється «дзеркало» даних. При виході з ладу одного з дисків вся інформація залишається збереженою на другому.

За такий стовідсотковий захист доводиться дорого платити – один вінчестер у вас працює просто так, не збільшуючи доступну ємкість ні на Мегабайт. При цьому немає ніякого виграшу в продуктивності.

Дуплекс – розвиток ідеї дзеркалювання. В цьому випадку маємо так само високий рівень надійності і потрібно в два рази більше жорстких дисків. Але з'являються додаткові витрати: для підвищення надійності в систему встановлюються два незалежних RAID контролери (рис.2.40). Вихід з ладу одного диска або контроллера не позначається на працездатності системи.

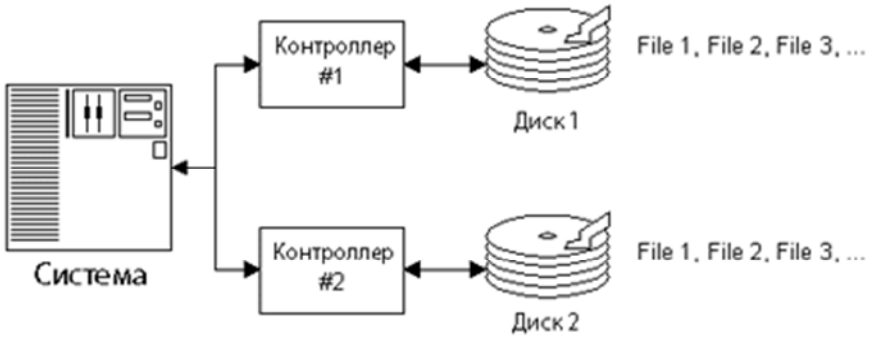


Рис.2.40 Технологія дуплексування

Таке дороге рішення використовується тільки в зовнішніх RAID-масивах, призначених для відповідальних застосувань.

Чергування – відмінна можливість підвищити швидкодію системи. Очевидно, якщо читання і запис вести паралельно на декількох жорстких дисках, можна отримати виграш в швидкості. Як це робиться? Записуваний файл розбивається на частини певного розміру і посилається одночасно на всі наявні накопичувачі (рис.2.41) . У такому фрагментованому вигляді файл і зберігається. Зчитується він теж «по шматочках».

Розмір «шматочка» може бути мінімальним – 1 байт, але частіше використовують крупніше ділення – по 512 байт (розмір сектора).

Парність є альтернативним рішенням, що поєднує в собі переваги дзеркалювання (висока надійність) і чергування (висока швидкість роботи). Використовується той же принцип, що і в контролі парності оперативної пам’яті.

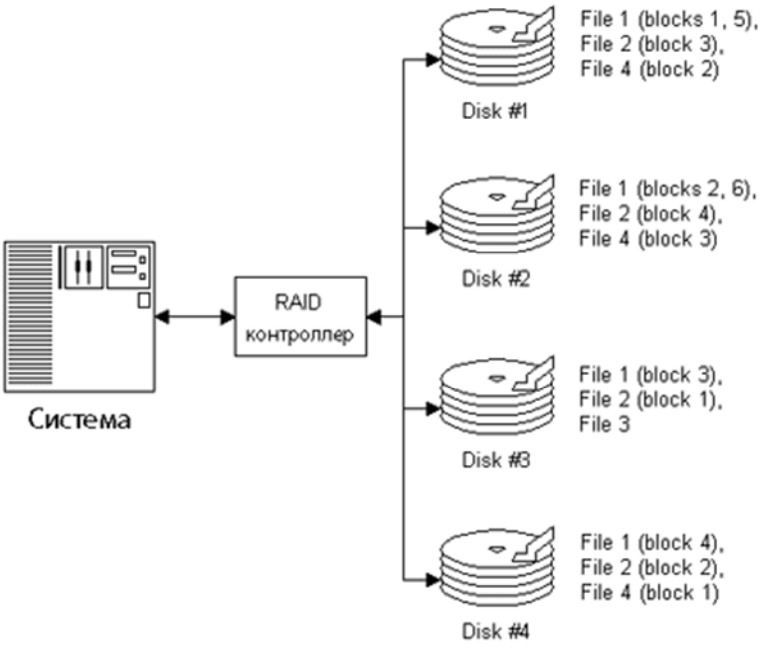


Рис. 2.41 Технологія чергування

Якщо є *N* блоків даних і на їх основі обчислюється ще один додатковий екстраблок, з отриманих *N*+1 блоків завжди можна відновити інформацію навіть при пошкодженні одного з них. Відповідно, для створення нормального RAID-масива в цьому випадку потрібно *N*+1 жорсткий диск.

Розподіл блоків по дисках такий самий, як при чергуванні. Екстраблок може записуватися на окремий накопичувач, або розкидатися по дисках.

Що ж зберігається в екстраблоці? Зазвичай кожен біт екстраблоку складається з суми біт всіх *N* блоків, точніше з результату виконання логічної операції XOR. Операція XOR має таку властивість, що при її повторному накладенні ми можемо отримати первинний результат. Тобто

$$(A \text{ XOR } B) \text{ XOR } B = A.$$

Це правило розповсюджується на будь-яку кількість операндів.

Плюси парності очевидні. За рахунок використання чергування підвищується швидкість роботи. При дзеркалюванні надійність зберігається,

але при цьому «неробочий» об’єм масиву помітно зменшується, він однаковий при будь-якій кількості дисків і складає ємкість одного диска, тобто при 5 дисках в масиві пропадає всього 20% ємкості.

Але у парності є вагомий мінус. Для формування екстраблоків потрібні обчислення. Їх треба робити на льоту, причому з мільйонами, мільярдами біт. Якщо цю справу доручити центральному процесору, ми отримаємо дуже повільну систему. Необхідно використовувати досить дорогі плати з RAID-контролерами, які «беруть всі обчислення на себе». У разі виходу з ладу одного з дисків, процес відновлення буде не таким швидким, як при дзеркалюванні.

Для оптимального використання вищенаведених технологій в окремих задачах, RAID прийнято класифікувати на рівні.

Відразу обмовимося, що є прості (single) і складені (multiple) масиви RAID. Складені є поєднанням двох простих, так що почнемо з простих 8-ми рівнів.

Рівень RAID 0

Простий масив, що використовує чергування без парності (рис.2.42) . Вся вхідна інформація розбивається на блоки фіксованої довжини (наприклад, 16 кбайт) і розкидається на все наявні диски.

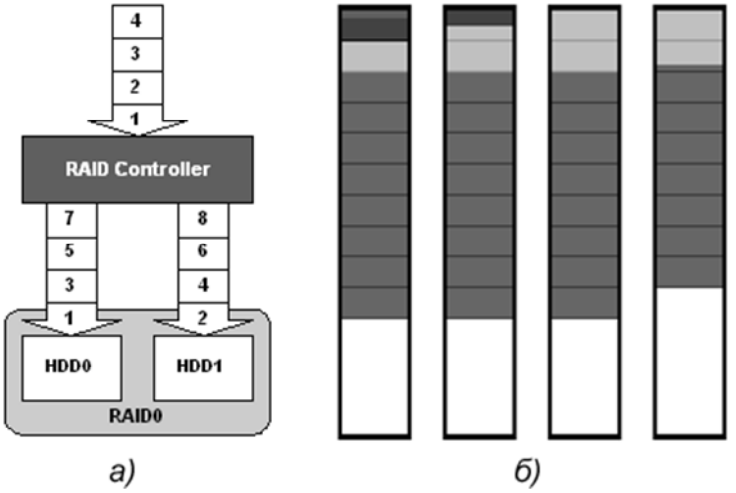


Рис. 2.42 RAID 0. *а*) - формування масивів, *б*) - розподіл даних

На малюнку приведений приклад, як в RAID 0 масиві на 4 дисках зберігаються дані різного розміру. Розмір блоку – 16 кбайт. Червоний – файл розміром 4 кбайт, синій, – 20 кбайт, зелений, – 100 кбайт, пурпурний, – 500 кбайт.

За наявності двох-чотирьох дисків RAID 0 дає відчутний виграш в швидкості передачі даних, але абсолютно не забезпечує надійність. Для його побудови підійде будь-який дешевий і навіть програмний RAID-контроллер. Підходить для тих, кому потрібно вижати максимум продуктивності від файлової системи при мінімальних витратах.

Рівень RAID 1

Цей рівень є звичайним дзеркалюванням (рис.2.43) . На два жорсткі диски пишуться дві однакові копії даних. При цьому можна використовувати дешевий контроллер або навіть його програмну реалізацію RAID.

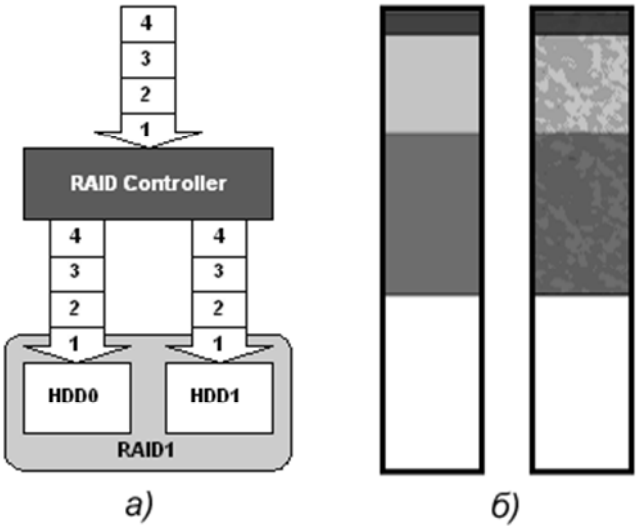


Рис. 2.43 RAID 1: *а*) - формування масивів, б) - розподіл даних

RAID 1 дозволяє надійно захистити дані і забезпечити роботу системи навіть при поломці одного з дисків. От чому він набув широкого поширення серед користувачів, охочих захистити від втрати особисті дані. Виграшу в швидкості при використанні RAID 1 немає.

Рівень RAID 2

Другий рівень RAID помер, так і не народившись. Все ті ж умільці з Берклі запропонували використовувати одночасно дві технології – побітове чергування і код Хеммінга для відновлення помилок (рис.2.44) . Теоретично це повинен бути непоганий по надійності і робочій ємкості масив, побудований з 14 або 39 дисків (!). Частина дисків (10 або 32) використовується для зберігання даних з чергуванням, останні – для зберігання вирахуваних контрольних сум. Реалізація таких систем вимагала спеціальних дорогих контроллерів, які так і не прижилися на ринку. У результаті RAID 2 зараз не використовується.

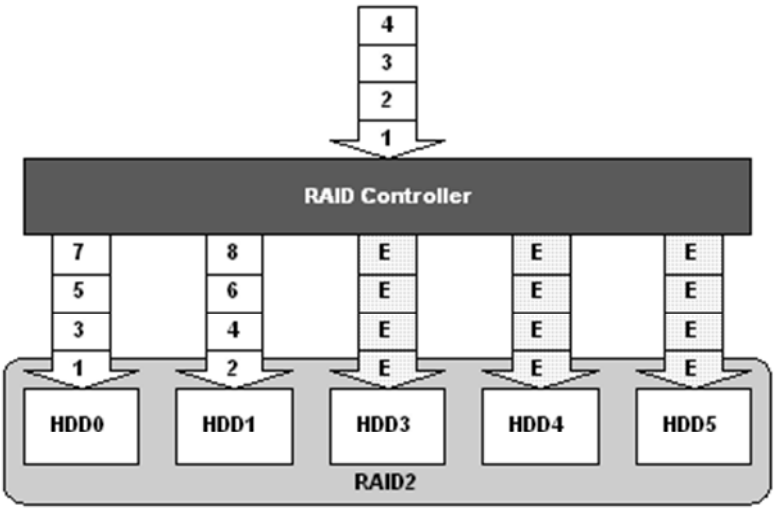


Рис. 2.44 Технологія RAID 2

Рівень RAID 3

Третій рівень використовує чергування і виділений диск для контролю парності (рис.3.16) . Блоки даних зазвичай мають довжину менше 1024 байт. Інформація розподіляється на декілька дисків, а вирахуване значення по парності зберігається на окремий диск.

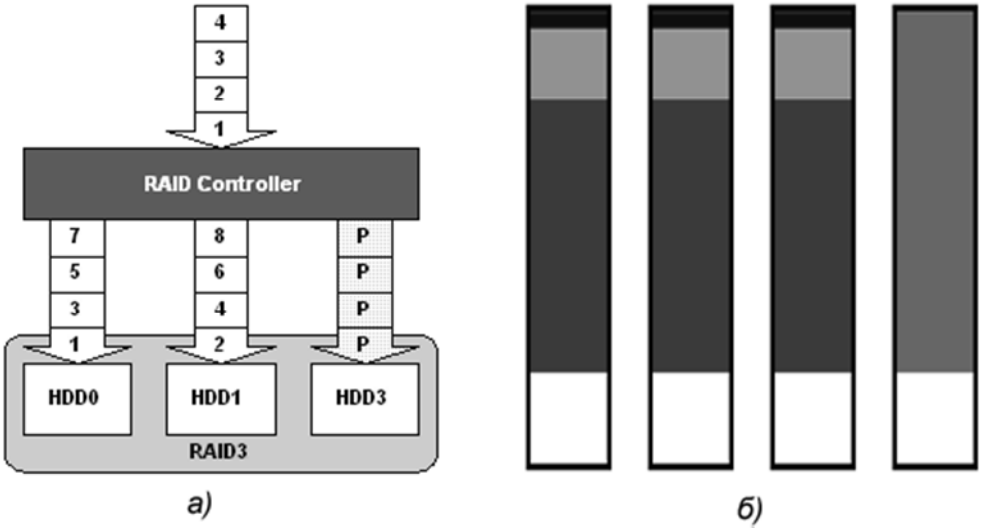


Рис. 2.45 RAID 3: а) - формування масивів, б) – розподіл даних

Всі швидкісні переваги чергування зводяться нанівець необхідністю записувати контрольну суму на виділений диск, а більше всього збільшується швидкість випадкового запису. До переваг віднесемо можливість роботи масиву при відмові одного з дисків.

Рівень RAID 4

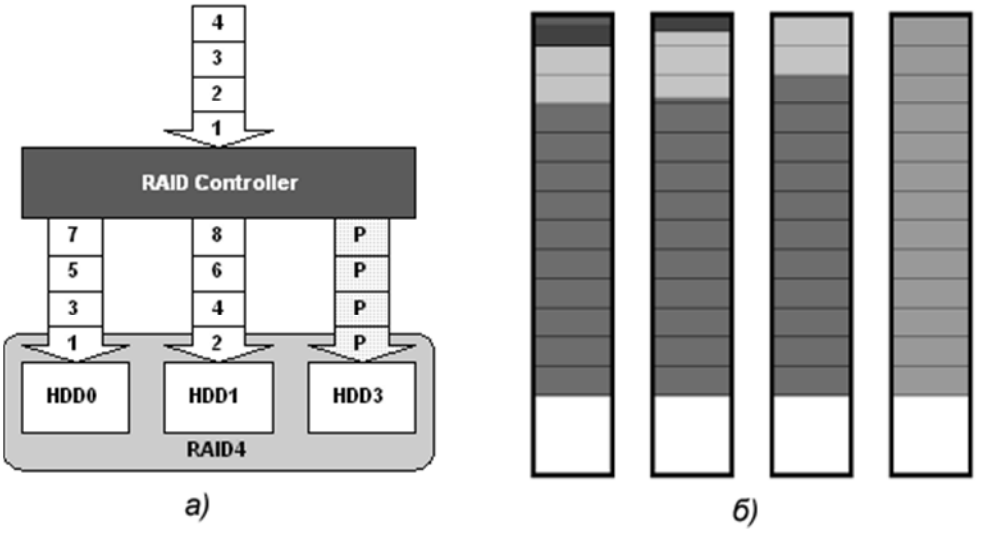


Рис. 2.46 RAID 4: *а*) - формування масивів, *б*) – розподіл даних

Відрізняється від RAID 3 тільки розміром блоку даних при чергуванні (рис.2.46) . Це декілька покращує роботу масиву при випадковому читанні,

але запис все одно досить повільна. Диск з контрольними сумами є яскраво вираженим «вузьким місцем» в системі.

Оскільки є компромісним варіантом між RAID 3 і RAID 5, не знайшов свого місця на ринку і рідко використовується. Це тримає ціни на відповідні контроллери на високому рівні.

Рівень RAID 5

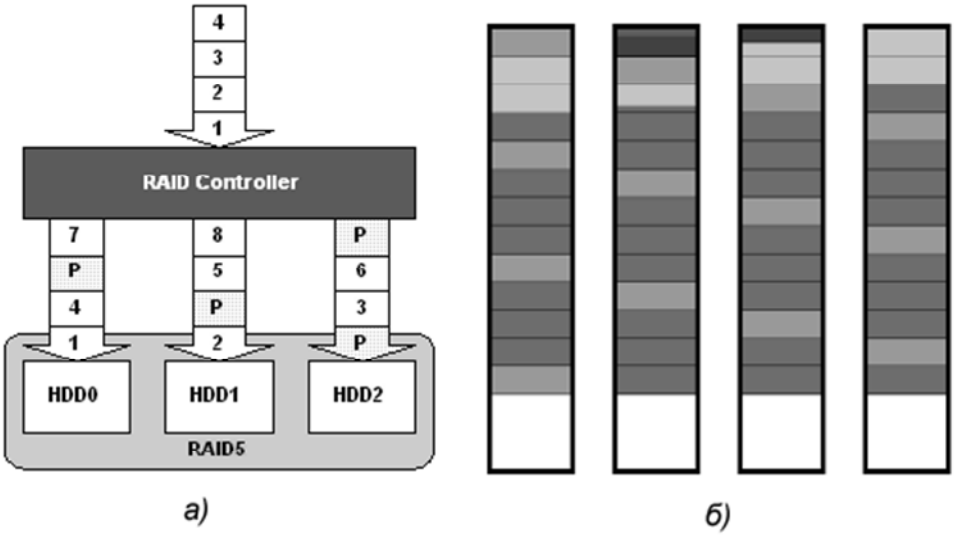


Рис. 2.47 RAID 5: *а*) - формування масивів, *б*) – розподіл даних

Найбільш поширений в системах зберігання даних – п’ятий рівень. Він характеризується застосуванням чергування і парності (рис.2.47) . На відміну від RAID 3, контрольні суми не зберігаються на одному диску, а розкидаються по всіх, що дозволяє значно підняти швидкість запису. Головний принцип розподілу екстраблоків: вони не повинні розташовуватися на тому ж диску, з якого була зашифрована інформація.

Надійність і швидкість роботи такої системи виявляються дуже навіть високими. При відновленні інформації всю роботу на себе бере RAID контроллер, так що операція проходить досить швидко.

Рівень RAID 6

Для деяких особливо критичних застосувань потрібна підвищена надійність. Наприклад, щоб при виході з ладу навіть двох дисків масив зберіг дані і навіть залишився працездатним. Чи можна це зробити? Звичайно, рішення лежить на поверхні.

Використовуються все ті ж технології чергування і парності. Але контрольна сума обчислюється двічі і копіюється на два різні диски. У результаті дані виявляться втраченими тільки у разі виходу з ладу відразу трьох жорстких дисків. В порівнянні з RAID 5 це дорожче і повільніше рішення, яке може показати себе хіба що при випадковому читанні. На практиці RAID 6 майже не використовується, оскільки вихід з ладу відразу двох дисків – дуже окремий випадок, а підвищити надійність можна іншими способами.

Рівень RAID 7

На відміну від решти рівнів, RAID 7 не є відкритим стандартом, таку звучну і вигідну назву вибрала для своєї модифікації RAID 3 компанія Storage Computer Corporation. Поліпшення полягають у використанні асинхронного чергування, застосуванні кеш-пам’яті і спеціального високопродуктивного мікропроцесора.

Забезпечуючи такий же, як в RAID 3, рівень надійності, RAID 7 значно виграє в швидкості. Недолік у нього один, але дуже серйозний – величезна ціна, обумовлена монополією на виготовлення контроллерів.

Складені рівні RAID

У основних рівнів RAID є свої переваги і недоліки. І цілком зрозуміло, чому інженери почали мріяти про таке RAID, який би об’єднував переваги декількох рівнів. Складений масив RAID – це найчастіше поєднання швидкого RAID 0 з надійним RAID 1, 3 або 5. Підсумковий масив дійсно

володіє покращуваними характеристиками, але і платити за це доводиться підвищенням вартості і складністю рішення.

Складений RAID будується так: спочатку диски розділяються на набори (set). Потім на основі кожного з наборів будуються прості масиви. А завершується все об’єднанням цих масивів в один мегамассив. Запис типу X+Y означає, що спочатку диски об’єднані в RAID рівня X, а потім декілька RAID X масивів об’єднані в RAID рівня Y.

RAID 0+1 (01) і 1+0 (10)

RAID 0+1 часто називають «дзеркалом страйпів», а RAID 1+0 – «страйпом дзеркал» (нормальне російське «чергування» практично не використовується, змінившись американізмом). У обох випадках використовуються дві технології – чергування і зеркалювання (рис.2.48), але результати різні.

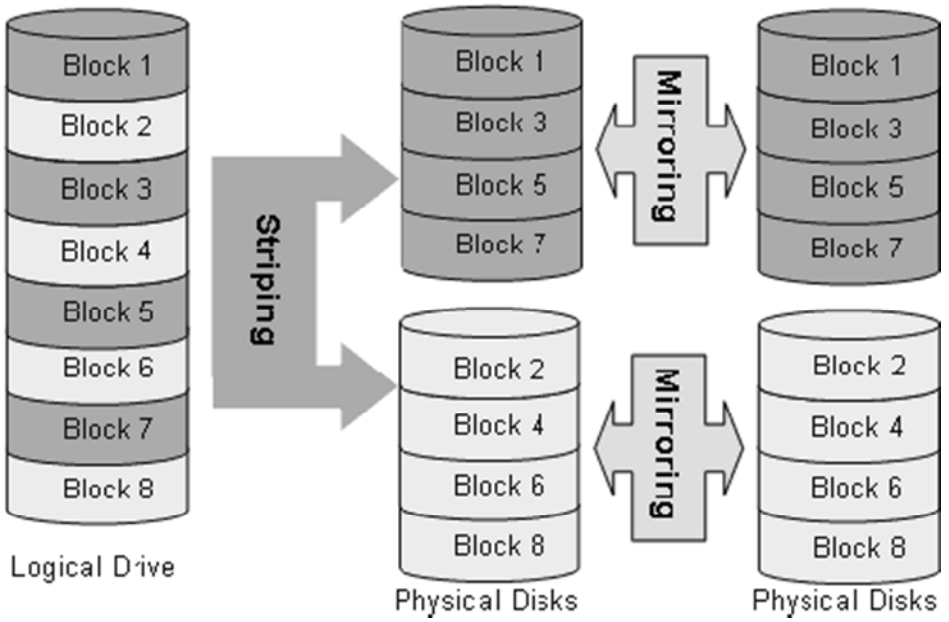


Рис. 2.48 Структура RAID 0+1

RAID 0+1 володіє високою швидкістю роботи і підвищеною надійністю, підтримується навіть дешевими контроллерами RAID і є недорогим рішенням. Але по надійності дещо краще RAID 1+0. Так, масив з 10 дисків (5 по 2) може залишитися працездатні пі відмові до 5 жорстких дисків!

Основний недолік цих масивів – низький відсоток використання ємкості накопичувачів – всього 50%. Але для домашніх систем саме RAID 01 або 10 може виявитися оптимальним рішенням (рис.2.49) .

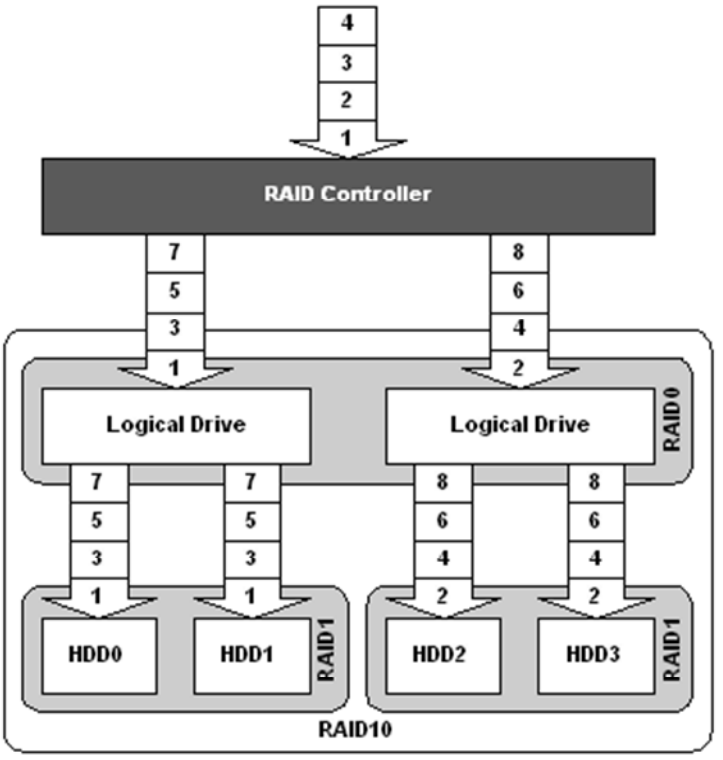


Рис. 2.49 Структура RAID 1+0

3 із рівнями RAID 0+3 (03) і 3+0 у виробників спостерігається плутанина. Досить часто замість 0+3 або 3+0 указують привабливіше число 5+3 (53). Не вірте!

По ідеї поєднання чергування і RAID 3 дає виграш в швидкості, але він досить малий. Зате система помітно ускладнюється. Найбільш простій рівень 3+0. З двох масивів RAID 3 будується страйп, і мінімальна кількість необхідних дисків – 6. RAID 3+0 з точки зору надійності краще ніж 0+3.

Переваги цих комбінацій в досить високому відсотку використання ємкості дисків і високої швидкості читання даних. Недоліки – висока ціна, складність системи.

Рівні RAID 0+5 (05) і 5+0 (50). Що буде, якщо об’єднати чергування з розподіленою парністю із звичайним чергуванням? Вийде швидка і надійна

система. RAID 0+5 є набором страйпов, на основі яких побудований RAID 5. Така комбінація використовується рідко, оскільки практично не дає виграшу ні в чому. Широкого поширення набув складений масив RAID 5+0 (рис.2.50).

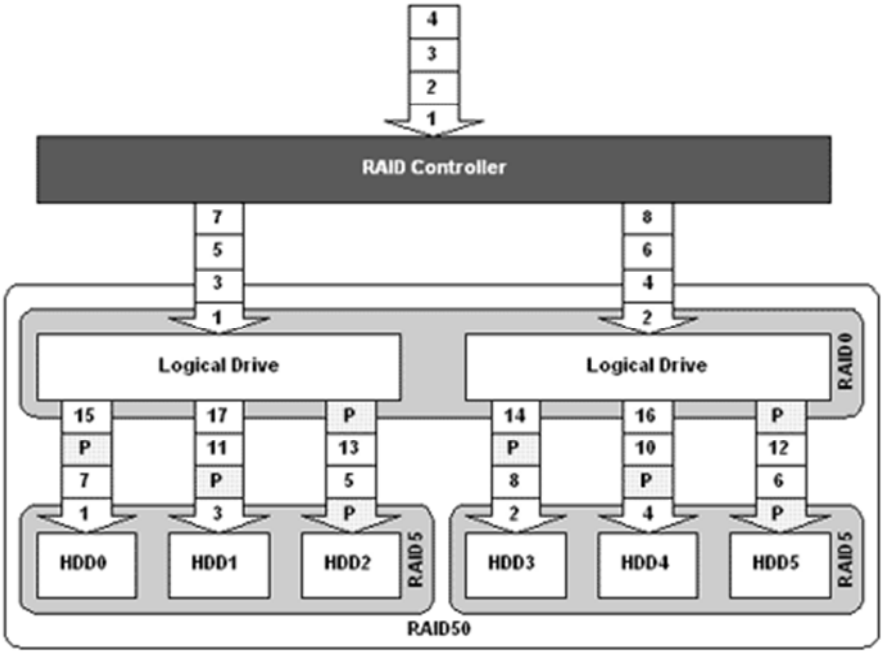


Рис. 2.50 Структура RAID 5+0

Найчастіше це два масиви RAID 5, об'єднаних в страйп. Така конфігурація дозволяє отримати високу продуктивність при роботі з файлами малого розміру. Типовий приклад – використання як WEB-сервера.

RAID 1+5 (15) і 5+1 (51)

Цей рівень побудований на поєднанні дзеркалювання або дуплексу і чергування з розподіленою парністю. Основна мета RAID 15 і 51 – значне підвищення надійності. Масив 1+5 продовжує працювати при відмові трьох накопичувачів, а 5+1 - навіть при втраті п'яти з восьми жорстких дисків! Платити доводиться великою кількістю невживаної ємкості дисків і загальним дорожчанням системи.

Найчастіше для побудови RAID 5+1 використовують два контроллери RAID 5, які зеркалируют на програмному рівні, що дозволяє понизити витрати.

Порівняння ефективності RAID технологій

В таблиці 2.3 наведено порівняння простих RAID-рівнів, а в таблиці 2.4 порівняння складених RAID-рівнів. В таблицях наведені основні параметри, які можуть бути корисними при виборі конкретної технології для кожної окремої задачі.

Таблиця 2.3 - Порівняння простих RAID-рівнів

	RAID 0	RAID 1	RAID 3	RAID 5	RAID 6
Технологія	Чергування	Дзеркалювання	Чергування, парність	Чергування, парність	Чергування, парність
Контроллер	Програмний	Програмний	Апаратний	Апаратний Hi-End	Спеціалізований
К-ть жорстких дисків	2, 4	2	3 і більше	3 і більше	3 і більше
Доступний робочий простір %	100	50	66 для 3, 75 для 4	66 для 3, 75 для 4	33 для 3 50 для 4 60 для 5
Стійкість при відмові диска	Немає	Висока	Висока	Висока	Дуже висока
Відновлення даних	Немає	Дуже швидке	Швидке	Швидке	Дуже швидке
Швидкість випадкового читання	Дуже хороша	Хороша	Хороша	Дуже хороша	Дуже хороша
Швидкість випадкового запису	Дуже хороша	Хороша	Погана	Нормальна	Погана
Швидкість лінійного читання	Дуже хороша	Хороша	Дуже хороша	Дуже хороша	Хороша
Швидкість лінійного запису	Дуже хороша	Хороша	Хороша	Хороша	Середня
Ціна	Найнижча	Низька	Середня	Середня	Висока

Технологія JBOD

А що робити, якщо потрібний просто один логічний диск гігантського розміру? Без всяких зеркалирований, чергування і парності? Тоді це вже не RAID, а JBOD – Just A Bunch Of Disks. Реалізувати цей режим здатний простий контроллер або навіть програмна реалізація контроллера (рис.2.51) .

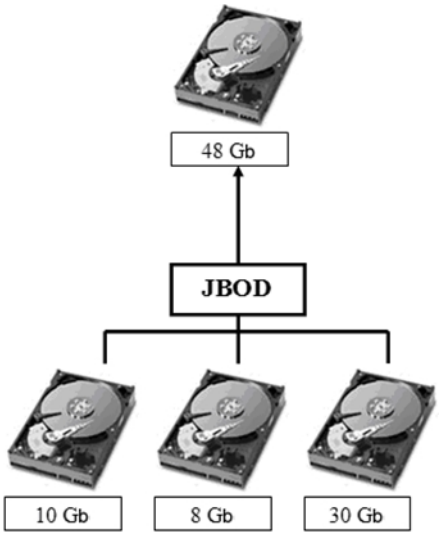


Рис. 2.51 Структура JBOD

Чи є у нього переваги, якщо JBOD не підвищує ні швидкодії, ні надійності? Є. Принаймні, для роботи використовується весь доступний простір жорстких дисків. І ще: у разі виходу з ладу одного з жорстких дисків, інформація на інших не ушкоджується.

Таблиця 2.4 - Порівняння складених RAID-рівнів

	RAID 0+1	**RAID 1+0**	**RAID 5+0**	**RAID 5+1**
Технологія	Чергування, дзеркалювання	Чергування, дзеркалювання	Чергування, парність	Чергування, парність, дзеркалювання
Контроллер	Програмний	Майже все	Спеціалізований	Спеціалізований
К-ть жорстких дисків	4 min	4 min	6 min	6 min
Доступний робочий простір %	50	50	66 для 2 страйпів по 3 диски	33-40
Стійкість при відмові диска	Дуже хороша	Відмінна	Хороша	Відмінна
Відновлення даних	Швидке	Дуже швидке	Середнє	Швидке
Швидкість випадкового читання	Дуже хороша	Дуже хороша	Дуже хороша	Дуже хороша
Швидкість випадкового запису	Хороша	Хороша	Хороша	Хороша
Швидкість лінійного читання	Дуже хороша	Дуже хороша	Дуже хороша	Дуже хороша
Швидкість лінійного запису	Хороша	Хороша	Хороша	Хороша
Ціна	Відносно висока	Відносно висока	Висока	Дуже висока

Контрольні питання до розділу 2

1. Комп’ютерні системи класу MIMD. SMP-системи, їх загальна архітектура, переваги і недоліки.
2. Аналіз видів архітектур SMP-систем.
3. Кластерні обчислювальні системи, їх особливості і недоліки.
4. Класифікація архітектур кластерних систем. Переваги і недоліки.
5. Комп’ютерні системи з масовою паралельною обробкою інформації.
6. Комп’ютерні системи з неоднорідним доступом до пам’яті.
7. Топології комп’ютерних систем. Загальні поняття.
8. Методи опису характеристик мережевих з’єднань.
9. Функції маршрутизації даних в КС. Перестановка, тасування, батерфляй.
10. Функції маршрутизації даних в КС. Реверсування бітів, зсув, мережа ILLIAC.
11. Статичні топології КС: лінійна, кільцева, зіркоподібна. Параметри, переваги і недоліки.
12. Статичні топології КС: деревоподібна, решітчата і повнозв’язна. Параметри, переваги і недоліки.
13. Статичні топології КС: тороїдальна, циліндрична і топологія гіперкуба. Параметри, переваги і недоліки.
14. Відмовостійкі комп’ютерні системи. Основні поняття.
15. Основні принципи забезпечення відмовостійкості дискової пам’яті. Технологія RAID0.
16. Технології RAID1, RAID2, RAID3 та порівняння їх ефективності.
17. Технології RAID4, RAID5. Порівняння їх ефективності.
18. Технології RAID6, RAID7. Порівняння їх ефективності.
19. Складені рівні RAID0+1 та RAID3+0.
20. Складені рівні RAID0+1 та RAID3+0.
21. Складені рівні RAID5+0 та RAID5+1. Технологія JBOD.

РОЗДІЛ 3. ОРГАНІЗАЦІЯ ОБЧИСЛЕНЬ В ПРКС

Тема 3.1. Операційні системи КС

3.1.1 Основні поняття операційних систем

Операційна система - сукупність програмних засобів, що забезпечує управління апаратною частиною комп'ютера і прикладними програмами, а також їх взаємодію між собою і користувачем.

ОС утворює автономну середу, не пов'язану ні з однією з мов програмування. Будь-яка ж прикладна програма пов'язана з операційною системою і може експлуатуватися тільки на тих комп'ютерах, де є аналогічна системна середовище. Прикладні програмні засоби, розроблені в середовищі однієї операційної системи, не можуть бути використані для роботи в середовищі іншої операційної системи, якщо немає спеціального комплексу програм (конвертера), що дозволяє це зробити. У такому випадку говорять про програмну несумісності комп'ютерів.

Сучасна комп'ютерна система складається з одного або декількох процесорів, оперативної пам'яті, дисків, клавіатури, монітора, принтерів, мережевого інтерфейсу та інших пристроїв, тобто є складною комплексною системою. Написання програм, які стежать за всіма компонентами, коректно використовують їх і при цьому працюють оптимально, являє собою вкрай важке завдання. З цієї причини комп'ютери оснащуються спеціальним рівнем програмного забезпечення, званим операційною системою. ОС відповідає за управління всіма перерахованими пристроями і забезпечує користувача програмами, що мають простий, доступний інтерфейс для роботи з апаратурою. Склад комп'ютерної системи умовно можна розбити на три

Склад операційної системи:

1. Програмний модуль, керуючий файлами.
2. Командний процесор (виконує команди користувача).

3. Програми, що забезпечують управління роботою різних пристроїв (введення, виведення, зберігання).

4. Графічний модуль.

5. Довідкова система.

6. Програми.

Етапи завантаження операційної системи:

1. Включення ПК.

2. Процесор звертається до ПЗУ за командами початкового завантаження.

3. Тестування підключених пристроїв.

4. Вивід характеристик знайдених пристроїв.

5. Завантажується операційна система із зовнішньої пам’яті (HDD) в оперативну.

6. Операційна система готова приймати завдання від користувача (Робочий Стіл).

Програми операційної системи, призначені для роботи під управлінням даної системи.

Основне призначення ОС - є завантаження прикладних програм і надання їм деяких сервісів.

Основна функція всіх ОС - посередницька. Вона полягає в забезпеченні декількох видів інтерфейсу:

- Інтерфейсу користувача та програмно-апаратними засобами комп’ютера (інтерфейс користувача);

- Інтерфейсу між програмним і апаратним забезпеченням (апаратно-програмний інтерфейс);

- Інтерфейсу між різними видами програмного забезпечення (програмний інтерфейс).

Навіть для однієї апаратної платформи IBM PC, існує декілька операційних систем. Відмінності між ними розглядають у двох категоріях: внутрішніх і зовнішніх.

Внутрішні відмінності характеризуються методами реалізації основних функцій.

Зовнішні відмінності визначаються наявністю і доступністю додатків даної системи, необхідних для задоволення технічних вимог, що пред’являються до конкретного робочого місця.

Всі ОС забезпечують свій автоматичний запуск. В спеціальній (системній) області диска створюється запис програмного коду. Звернення до цього коду виконують програми, що знаходяться в BIOS. Завершуючи свою роботу, вони дають команду на завантаження і виконання вмісту системної області диска.

Всі сучасні ОС забезпечують створення файлової системи, призначеної для зберігання даних на дисках і забезпечення доступу до них.

До функції обслуговування файлової структури належать такі операції, що відбуваються під управлінням ОС:

- Створення файлів і присвоєння їм імен;

- Створення каталогів (папок) і присвоєння їм імен;

- Перейменування файлів і каталогів (папок);

- Копіювання і переміщення файлів між дисками комп’ютера і між каталогами (папками) одного диска;

- Видалення файлів і каталогів (папок);

- Навігація по файловій структурі з метою доступу до заданого файлу, каталогу (папки);

- Управління атрибутами файлів і каталогів (папок).

ОС можна класифікувати різними способами, розглянемо один з них - за функціональним призначенням.

3.1.2 Види операційних систем

Дискові Операційні Системи.

Це системи, що беруть на себе виконання тільки чотирьох функцій:

- забезпечувати завантаження призначених для користувача програм в оперативну пам’ять і їх виконання;
- забезпечувати управління пам’яттю, в багатопроцесорних системах це складне завдання управління системними ресурсами;
- забезпечувати роботу з пристроями довгострокової пам’яті, такими як магнітні диски, оптичні диски, флеш-пам’ять і т. д. як правило, ОС управляє вільним простором на цих носіях і структурує дані користувача у вигляді файлових систем;
- надавати більш-менш стандартизований доступ до різних периферійних пристроїв, таких як термінали, модеми, друкуючі пристрої.

ОС загального призначення.

Тут під ОС маються на увазі системи «загального призначення», тобто розраховані на інтерактивну роботу одного або декількох користувачів в режимі поділу часу, при не дуже жорстких вимогах до часу реакції системи на зовнішні події. Як правило, в таких системах приділяється велика увага захисту самої системи, програмного забезпечення і призначених для користувача даних від помилкових і зловмисних програм і користувачів. Зазвичай подібні системи використовують вбудовані в архітектуру процесора засоби захисту і віртуалізації пам’яті. До цього класу належать такі широко розповсюджені системи, як Windows, системи сімейства Unix.

ОС реального часу.

Це системи, призначені для полегшення розробки так званих додатків реального часу - програм, керівників некомп’ютерним обладнанням, часто з дуже жорсткими обмеженнями за часом. Прикладом такого застосування може бути програма бортового комп’ютера fly-by-wire (дослівно - «летить по дроті», тобто використовує систему управління, в якій органи управління не мають механічного і гідравлічного зв’язку з керуючими органами) літака,

системи управління прискорювачем елементарних частинок або промисловим обладнанням. Подібні системи зобов'язані підтримувати багатопоточність, гарантований час реакції на зовнішню подію, простий доступ до таймера та периферійного обладнання.

ОС проміжних типів.

Існують системи, які не можна віднести до одного з перерахованих вище класів. Така, наприклад, система RT-11, яка, по суті своїй, є ДОС, але дозволяє одночасне виконання декількох програм з досить багатими засобами взаємодії і синхронізації. Іншим прикладом проміжної системи є MS Windows 3.x і Windows 95, які, як ОС, використовують апаратні засоби процесора для захисту і віртуалізації пам'яті і навіть можуть забезпечувати певна подібність багатозадачності, але не захищають себе і програми від помилок інших програм, подібно ДОС.

Системи віртуальних машин.

Такі системи стоять дещо осібно. Система віртуальних машин - це ОС, що допускає одночасну роботу декількох програм, але створює при цьому для кожної програми ілюзію того, що машина знаходиться в повному її розпорядженні, як при роботі під управлінням ДОС. Найчастіше, «програмою» виявляється повноцінна операційна система - прикладами таких систем є VMWare для машин з архітектурою ix86 або VM для System/370 та її нащадків.

Віртуальні машини є цінним засобом при розробці та тестуванні крос-платформних додатків. Рідше вони використовуються для налагодження модулів ядра або самої ОС.

Такі системи відрізняються високими накладними витратами і порівняно низькою надійністю, тому відносно рідко знаходять промислове застосування.

Поняття **«обчислювальний процес»** (або просто - **«процес»)** є одним з основних при розгляді ОС. Як поняття, процес є певним видом абстракції. Послідовний процес (іноді званий «завданням») - це виконання окремої

програми з її даними на послідовному процесорі. Концептуально процесор розглядається у двох аспектах:

- він є носієм даних;
- він (одночасно) виконує операції, пов’язані з їх обробкою.

В якості прикладів можна назвати наступні процеси (завдання):

- виконання прикладних програм користувачів;
- виконання утиліт;
- виконання інших системних обробних програм;
- редагування якого-небудь тексту;
- трансляція вихідної програми;
- компонування вихідної програми;
- виконання вихідної програми.

Ресурси можуть розділяються, коли декілька процесів можуть їх використовувати одночасно (в один і той же момент часу) або паралельно (протягом деякого інтервалу часу процеси використовують ресурс поперемінно), а можуть бути і неподільними

Переривання являють собою механізм, що дозволяє координувати паралельне функціонування окремих пристроїв обчислювальної системи і реагувати на особливі стани, що виникають при роботі процесора. Таким чином, переривання - це примусова передача керування від виконуваної програми до системи (а через неї - до відповідної програмі обробки переривання), що відбувається при виникненні певної події.

Механізм переривань реалізується апаратно-програмними засобами. Структури систем переривання (у залежності від апаратної архітектури) можуть бути самими різними, але всі вони мають одну загальну особливість - переривання неодмінно тягне за собою зміну порядку виконання команд процесором.

Механізм обробки переривань незалежно від архітектури обчислювальної системи включає наступні елементи:

1. Встановлення факту переривання (прийом сигналу на переривання) і ідентифікація переривання (в операційних системах іноді здійснюється повторно, на кроці 4).

2. Запам’ятовування стану перерваного процесу. Стан процесу визначається, перш за все, значенням лічильника команд (адресою наступної команди), вмістом регістрів процесора і може включати також специфікацію режиму (наприклад, режим призначений для користувача або привілейований) та іншу інформацію.

3. Управління апаратно передається підпрограмі обробки переривання. У найпростішому випадку в лічильник команд заноситься початкова адреса підпрограми обробки переривань, а у відповідні регістри - інформація зі слова стану.

4. Збереження інформації про перервану програму, яку не вдалося врятувати на кроці 2 за допомогою дій апаратури. У деяких обчислювальних системах передбачається запам’ятовування досить великого обсягу інформації про стан перерваного процесу.

5. Обробка переривання. Ця робота може бути виконана тією ж підпрограмою, якій було передано управління на кроці 3, але в ОС найчастіше вона реалізується шляхом наступного виклику відповідної підпрограми.

6. Відновлення інформації, що відноситься до перерваного процесу (етап, зворотний кроку 4).

7. Повернення в перервану програму.

Кроки 1-3 реалізуються апаратно, а кроки 4-7 - програмно.

Тема 3.2. Механізми взаємодії процесів

3.2.1 Основи взаємодії процесів.

Головною особливістю взаємодії є простота технічної реалізації обміну даними між процесами — усі потоки одного процесу використовують один адресний простір, а отже, можуть вільно отримувати доступ до спільно використовуваних даних, ніби вони є їх власними. Оскільки технічних труднощів із реалізацією обміну даними тут немає, основною проблемою, яку потрібно вирішувати в цьому випадку, є синхронізація потоків.

Для розв’язання задач синхронізації потоків одного процесу можна використовувати різні синхронізаційні примітиви. До найпростіших примітивів належать м’ютекси, семафори, умовні змінні, блокування читання-записування і бар’єри.

Механізмом більш високого рівня є концепція монітора, що поєднує м’ютекси та умовні змінні, а також задає деякі правила їхньої взаємодії для захисту спільно використовуваних даних.

З іншого боку, кожен потік виконується в рамках адресного простору деякого процесу, тому часто постає задача організації взаємодії між потоками різних процесів (рис. 3.1).

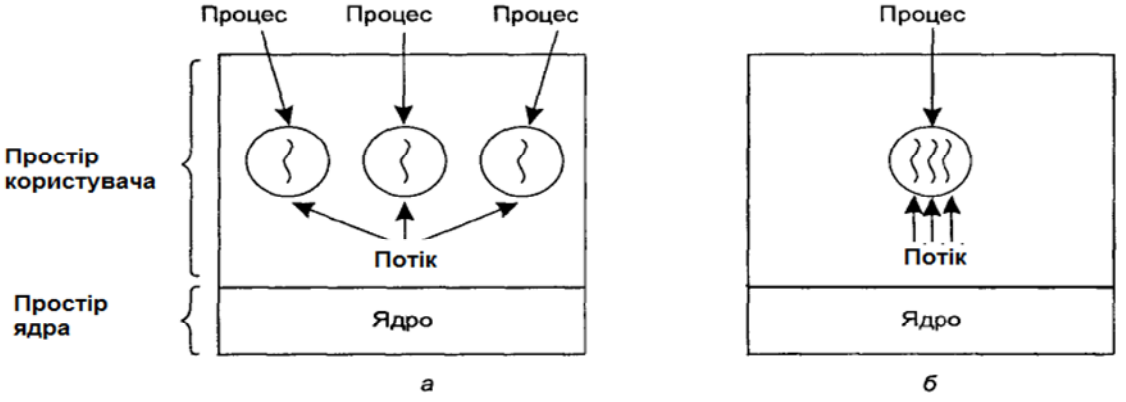


Рис. 3.1. а) три процеси з одиночними потоками
б) один процес з трьома потоками.

Для потоків різних процесів питання забезпечення синхронізації теж є актуальними, але вони в більшості випадків не грунтуються на понятті

спільно використовуваних даних (такі дані за замовчуванням для процесів відсутні). Крім того, додається досить складна задача забезпечення обміну даними між захищеними адресними просторами. Підходи до її розв’язання визначають різні види міжпроцесової взаємодії.

Реалізація міжпроцесової взаємодії здійснюється трьома основними методами: передавання повідомлень, розподілюваної пам’яті та відображуваної пам’яті.

Методи розподілюваної пам’яті (shared memory) дають змогу двом процесам обмінюватися даними через загальний буфер пам’яті. Перед обміном даними кожний із процесів має приєднати цей буфер до свого адресного простору з використанням спеціальних системних викликів (перед цим перевіряють права). Буфер доступний у системі за допомогою процедури іменування, термін його існування звичайно обмежений часом роботи всієї системи. Дані в ньому фактично є спільно використовуваними, як і для потоків. Жодних засобів синхронізації доступу до цих даних розподілювана пам’ять не забезпечує, програміст, так само, як і при розробці багатопотокових застосунків, має організовувати її сам.

В основі методів передавання повідомлень (message passing) лежать різні технології, що дають змогу потокам різних процесів (які, можливо, виконуються на різних комп’ютерах) обмінюватися інформацією у вигляді фрагментів даних фіксованої чи змінної довжини, котрі називають повідомленнями (messages). Процеси можуть приймати і відсилати повідомлення, при цьому автоматично забезпечується їхнє пересилання між адресними просторами процесів одного комп’ютера або через мережу. Важливою особливістю технологій передавання повідомлень є те, що вони не спираються на спільно використовувані дані — процеси можуть обмінюватися повідомленнями, навіть не знаючи один про одного.

Технологія відображуваної пам’яті

Ще однією категорією засобів міжпроцесової взаємодії є відображувана пам’ять (mapped memory). У ряді ОС відображувана пам’ять є базовим системним механізмом, на якому грунтуються інші види міжпроцесової взаємодії та системні рішення. Звичайно відображувану пам’ять використовують у поєднанні з інтерфейсом файлової системи, в такому разі говорять про файли, відображувані у пам’ять (memory-mapped files).

Ця технологія зводиться до того, що за допомогою спеціального системного виклику певну частину адресного простору процесу однозначно пов’язують із вмістом файла. Після цього будь-яка операція записування в таку пам’ять спричиняє зміну вмісту відображеного файла, яка відразу стає доступною усім застосункам, що мають доступ до цього файла. Інші застосунки теж можуть відобразити той самий файл у свій адресний простір і обмінюватися через нього даними один з одним.

3.2.2 Особливості міжпроцесової взаємодії

Тепер можна порівняти характеристики міжпроцесової взаємодії із характеристиками взаємодії потоків одного процесу.

- Проблема організації обміну даними є актуальною тільки для міжпроцесової взаємодії, оскільки потоки обмінюються даними через загальний адресний простір. Обмін даними між потоками схожий на використання розподілюваної пам’яті, але не потребує підготовчих дій.

- Проблема синхронізації доступу до спільно використовуваних даних є актуальною для взаємодії потоків і для міжпроцесової взаємодії із використанням розподілюваної пам’яті. Використання механізму передавання повідомлень не грунтується на спільно використовуваних даних.

Базові механізми взаємодії процесів

Розглянемо особливості організації взаємодії між потоками різних процесів. Основною характеристикою такої взаємодії є те, що у процесів немає спільного адресного простору, тому тут не можна безпосередньо працювати зі спільно використовуваними даними, як це було можливо для потоків. Тут ітиметься переважно про процеси, під якими розуміють потоки різних процесів.

Міжпроцесова взаємодія на базі спільної пам’яті

Для вирішення проблеми міжпроцесової синхронізації необхідно:

- по-перше, організувати спільну пам’ять між процесами (це може бути розподілювана пам’ять або файл, відображений у пам’ять);

- по-друге, розмістити в цій пам’яті стандартні синхронізаційні об’єкти (семафори, м’ютекси, умовні змінні);

- по-третє, використовуючи ці об’єкти, працювати зі спільно використовуваними даними, як це робилося у разі використання потоків.

Такий підхід широко застосовують на практиці. На жаль, досить складно запропонувати спосіб його реалізації для міжпроцесової синхронізації у більшості систем, оскільки різні системи пропонують різний набір засобів організації спільної пам’яті та засобів сигналізації, які можуть працювати в такій пам’яті.

За допомогою відображуваних файлів можна легко реалізувати розподіл пам’яті між процесами, якщо відобразити один і той самий файл на адресний простір кількох із них.

Відображення файла у пам’ять можна використати для завантаження виконуваних файлів у пам’ять (такі файли відображаються в адресний простір процесу і є простором підтримки для сторінок коду). Цей підхід використовується у Windows XP і в Linux.

На основі відображуваної пам’яті можна також реалізувати кешування диска.

За допомогою цієї технології можна виділяти пам’ять процесу. Для цього в його адресний простір відображають спеціальний файл, у разі доступу до якого завжди повертають нулі.

Можна забезпечити автоматичне збереження значень складних структур даних між викликами програми (такі структури створюють у ділянці пам’яті, що відповідає відображеному файлу; під час наступних викликів цей файл відображають у ту саму ділянку пам’яті знову, і всі структури поновлюються в цій пам’яті повністю).

3.2.3 Основи передавання повідомлень

Усі методи взаємодії, які було розглянуто дотепер, грунтуються на читанні й записуванні спільно використовуваних даних. На практиці така взаємодія не завжди можлива (наприклад, робота зі спільно використовуваними даними проблематична, якщо для процесів немає спільної фізичної пам’яті, а є тільки мережний зв’язок між комп’ютерами, на яких вони виконуються). У таких випадках можна використати засоби взаємодії, які не грунтуються на спільно використовуваних даних, передусім засоби передавання повідомлень.

Як було вже згадано, засоби передавання повідомлень грунтуються на обміні повідомленнями — фрагментами даних змінної довжини. Основою такого обміну є не спільна пам’ять, а канал зв’язку (communication channel). Він забезпечує взаємодію між процесами (для того, щоб спілкуватися, вони повинні створити канал зв’язку) і є абстрактним відображенням мережі зв’язку. Абстрактність каналу дає змогу реалізувати його не тільки на основі мережної взаємодії, але й спільної пам’яті (коли процеси перебувають на одному комп’ютері). При цьому такі зміни в реалізації будуть приховані від процесів, що взаємодіють.

Виокремлюють такі характеристики каналів зв’язку: спосіб задання; кількість процесів, які можуть бути з’єднані одним каналом; кількість каналів, які можуть бути створені між двома процесами; пропускна здатність

каналу (кількість повідомлень, які можуть одночасно перебувати в системі й бути асоційованими з цим каналом); максимальний розмір повідомлення; спрямованість зв’язку через канал (двобічний або однобічний зв’язок).

Примітиви передавання повідомлень

Основна особливість передавання повідомлень полягає в тому, що процеси спільно використовують тільки канали. Немає необхідності забезпечувати взаємне виключення процесів під час доступу до спільно використовуваних даних, замість цього досить визначити примітиви передавання повідомлень — спеціальні операції обміну даними через канал, які забезпечують не лише обмін даними, але й синхронізацію.

Є два примітиви передавання повідомлень: **send** (для відсилання повідомлення каналом) і **receive** (для отримання повідомлення з каналу). Розглянемо, як особливості реалізації send і receive дають змогу виділити різні класи методів передавання повідомлень.

Зазначені примітиви передавання повідомлень можуть задавати прямий і непрямий обмін даними. При прямому обміні даними необхідно явно вказувати процес, з яким необхідно обмінюватись інформацією. Непрямий обмін здійснюють через спеціальний об’єкт (поштову скриньку, порт); процеси можуть поміщати повідомлення в поштову скриньку і отримувати їх звідти.

Синхронне й асинхронне передавання повідомлень

Зупинимося на основних питаннях синхронізації під час передавання повідомлень, можна виокремити різні групи методів передавання повідомлень залежно від того як вони дають можливість відповісти на два запитання. Чи може потік бути призупинений під час виконання операції send, якщо повідомлення не було отримане?

- Чи може потік бути призупинений під час виконання операції receive, якщо повідомлення не було відіслане?
- У реальних системах відповідь на друге запитання практично завжди буде позитивною — неблокувальне приймання повідомлень

призводить до того, що вони губляться. Варіанти відповідей на перше запитання визначають два класи передавання повідомлень — синхронне і асинхронне.

Під час синхронного передавання повідомлень операція send призупиняє процес до отримання повідомлення, а під час асинхронного передавання повідомлень - не призупиняє процес (тобто є неблокувальною); після відсилання повідомлення процес продовжує своє виконання, не чекаючи отримання результату.

Реалізація синхронного й асинхронного передавання повідомлень залежить від низки характеристик каналу й обміну повідомленнями, насамперед від пропускної здатності каналу.

Якщо пропускна здатність дорівнює нулю (повідомлення не можуть очікувати в системі), відправник завжди має очікувати, поки одержувачу не надійде повідомлення, а одержувач має очікувати, поки повідомлення йому не буде відіслано. Два процеси мають явно домовлятися про майбутній обмін.

Якщо пропускна здатність обмежена (у системі можуть перебувати максимум п повідомлень для цього каналу), відправник має очікувати тільки тоді, коли черга повідомлень для цього каналу переповнена (у ній перебуває рівно п повідомлень), одержувач має очікувати, якщо ця черга порожня.

Якщо пропускна здатність необмежена, очікування можливе тільки для одержувача за порожньої черги.

Під час обміну повідомленнями необхідне підтвердження їх отримання. Деякі методи обміну повідомленнями не вимагають підтвердження зовсім, в інших випадках можлива ситуація, коли відправника після виконання операції send блокують доти, поки одержувач не надішле йому інше повідомлення із підтвердженням отримання; таку технологію називають обміном повідомленнями із підтвердженням отримання.

3.2.4 Технології передавання повідомлень

Розглянемо методи передавання повідомлень, які застосовують на практиці.

Канал — це найпростіший засіб передавання повідомлень. Він є циклічним буфером, записування у який виконують за допомогою одного процесу, а читання — за допомогою іншого. У конкретний момент часу до каналу має доступ тільки один процес. Операційна система забезпечує синхронізацію згідно правилу: якщо процес намагається записувати в канал, у якому немає місця, або намагається зчитати більше даних, ніж поміщено в канал, він переходить у стан очікування.

Розрізняють безіменні та поіменовані канали.

До безіменних каналів немає доступу за допомогою засобів іменування, тому процес не може відкрити вже наявний безіменний канал без його дескриптора. Це означає, що такий процес має отримати дескриптор каналу від процесу, що його створив, а це можливо тільки для зв’язаних процесів.

До поіменованих каналів (named pipes) є доступ за іменем. Такому каналу може відповідати, наприклад, файл у файловій системі, при цьому будь-який процес, який має доступ до цього файла, може обмінюватися даними через відповідний канал. Поіменовані канали реалізують непрямий обмін даними.

Обмін даними через канал може бути однобічним і двобічним.

Іншою технологією асинхронного непрямого обміну даними є застосування черг повідомлень (message queues). Для таких черг виділяють спеціальне місце в системній ділянці пам’яті ОС, доступне для застосувань користувача. Процеси можуть створювати нові черги, відсилати повідомлення в конкретну чергу й отримувати їх звідти. Із чергою одночасно може працювати кілька процесів. Повідомлення — це структури даних змінної довжини. Для того щоб процеси могли розрізняти адресовані їм повідомлення, кожному з них присвоюють тип. Відіслане повідомлення

залишається в черзі доти, поки не буде зчитане. Синхронізація під час роботи з чергами схожа на синхронізацію для каналів.

Найрозповсюдженішим методом обміну повідомленнями є використання сокетів (sockets). Ця технологія насамперед призначена для організації мережного обміну даними, але може бути використана й для взаємодії між процесами на одному комп'ютері.

Сокет — це абстрактна кінцева точка з'єднання, через яку процес може відсилати або отримувати повідомлення. Обмін даними між двома процесами здійснюють через пару сокетів, по одному на кожен процес. Абстрактність сокету полягає в тому, що він приховує особливості реалізації передавання повідомлень — після того як сокет створений, робота з ним не залежить від технології передавання даних, тому один і той самий код можна без великих змін використовувати для роботи із різними протоколами зв'язку.

Особливості протоколу передавання даних і формування адреси сокету визначає комунікаційний домен; його потрібно зазначати під час створення кожного сокету. Прикладами доменів можуть бути домен Інтернету (який задає протокол зв'язку на базі TCP/IP) і локальний домен або домен UNIX, що реалізує зв'язок із використанням імені файла (подібно до поіменованого каналу). Сокет можна використовувати у поєднанні тільки з одним комунікаційним доменом. Адреса сокету залежить від домену (наприклад, для сокетів домену UNIX такою адресою буде ім'я файла).

Способи передавання даних через сокет визначаються його типом. У конкретному домені можуть підтримуватися або не підтримуватися різні типи сокетів

Наприклад, і для домену Інтернет, і для домену UNIX підтримуються сокети таких типів:

- потокові (stream sockets) — задають надійний двобічний обмін даними суцільним потоком без виділення меж (операція читання даних повертає стільки даних, скільки запитано або скільки було на цей момент передано);

- дейтаграмні (datagram sockets) — задають ненадійний двобічний обмін повідомленнями із виділенням меж (операція читання даних повертає розмір того повідомлення, яке було відіслано).

Під час обміну даними із використанням сокетів зазвичай застосовується технологія клієнт-сервер, коли один процес (сервер) очікує з'єднання, а інший (клієнт) з'єднують із ним.

Перед тим як почати працювати з сокетами, будь-який процес (і клієнт, і сервер) має створити сокет за допомогою системного виклику socket ( ). Параметрами цього виклику задають комунікаційний домен і тип сокету. Цей виклик повертає дескриптор сокету — унікальне значення, за яким можна буде звертатися до цього сокету.

Подальші дії відрізняються для сервера і клієнта. Спочатку розглянемо послідовність кроків, яку потрібно виконати для сервера.

1. Сокет пов'язують з адресою за допомогою системного виклику bind( )). Для сокетів домену UNIX як адресу задають ім'я файла, для сокетів домену Інтернету — необхідні характеристики мережного з'єднання. Далі клієнт для встановлення з'єднання й обміну повідомленнями має буде вказати цю адресу.

2. Сервер дає змогу клієнтам встановлювати з'єднання, виконавши системний виклик listen( ) для дескриптора сокету, створеного раніше.

3. Після виходу із системного виклику listen( ) сервер готовий приймати від клієнтів запити на з'єднання. Ці запити вишиковуються в чергу. Для отримання запиту із цієї черги і створення з'єднання використовують системний виклик accept( ). Внаслідок його виконання в застосування повертають новий сокет для обміну даними із клієнтом. Старий сокет можна використовувати далі для приймання нових запитів на з'єднання. Якщо під час виклику accept( ) запити на з'єднання в черзі відсутні, сервер переходить у стан очікування.

Для клієнта послідовність дій після створення сокету зовсім інша. Замість трьох кроків досить виконати один - встановити з'єднання із

використанням системного виклику connect( ). Параметрами цього виклику задають дескриптор створеного раніше сокету, а також адресу, подібну до вказаної на сервері для виклику bind( ).

Після встановлення з’єднання (і на клієнті, і на сервері) з’явиться можливість передавати і приймати дані з використанням цього з’єднання. Для передавання даних застосовують системний виклик send(), а для приймання — recv().

Зазначену послідовність кроків використовують для встановлення надійного з’єднання.

Технологія віддаленого виклику процедур (Remote Procedure Call, RPC) є прикладом синхронного обміну повідомленнями із підтвердженням отримання. Розглянемо послідовність кроків, необхідних для обміну даними в цьому разі.

1. Операцію send оформляють як виклик процедури із параметрами.

2. Після виклику такої процедури відправник переходить у стан очікування, а дані (ім’я процедури і параметри) доставляються одержувачеві. Одержувач може перебувати на тому самому комп’ютері, чи на віддаленій машині; технологія RPC приховує це. Класичний віддалений виклик процедур передбачає, що процес-одержувач створено внаслідок запиту.

3. Одержувач виконує операцію receive і на підставі даних, що надійшли, виконує відповідні дії (викликає локальну процедуру за іменем, передає їй параметри і обчислює результат).

4. Обчислений результат повертають відправникові як окреме повідомлення.

5. Після отримання цього повідомлення відправник продовжує своє виконання, розглядаючи обчислений результат як наслідок виклику процедури.

Тема 3.3. Паралельні алгоритми

3.3.1 Основи паралельних алгоритмів

Паралелізм – сукупність математичних, алгоритмічих, програмних і апаратних засобів, що забезпечують можливість пералельного виконання задачі.

Розподілені обчислення – сукупність протоколів обміну та незалежних апаратних засобів (комп'ютерів, серверів), що представляються користувачу єдиним обчислювачем, придатним для вирішення складної задачі

Є коло обчислювальних задач, що вимагає більших обчислювальних ресурсів, ніж надає ПЕОМ. Для задач необхідно:

- більша швидкодія;
- більший об'єм оперативної пам'яті;
- велике кількість інформації, що передається;
- обработка і зберігання великого об'єму інформації.

При наявності хоча б однієї з наведених вимог використання паралельної обробки оправдано.

Серед задач, під паралельну обробку найбільше підходять: розпізнавання і синтез мови, розпізнавання зображень. Наприклад:

1. Складні, багатовимірні задачі, які необхідно розв'язати на протязі досить обмеженого часу, вимагають забезпечення великої швидкодії, наприклад - задачі прогнозу погоди. Область розв'язку (атмосфера) розбивається на окремі просторові зони, причому для розв'язку часових змін обчислень в кожній зоні повторюється багато разів. Якщо об'єм зони рівний 1 км^3, то для моделювання 10 км шару атмосфери необхідно 5x10^8 таких зон. Припустимо, що обчислення в кожній зоні вимагає 200 операцій з плаваючою крапкою, тоді за один часовий крок необхідно виконати 10^{11} операцій з плаваючою крапкою. Для того, щоб провести розрахунок прогнозу погоди з передбачуванністю 10 днів з 10-ти хвилинним кроком в часу, ЕОМ продуктивністю 100 Mflops (10^8 операцій з плаваючою крапкою за секунду)

необхідно 10^7 секунд чи понад 100 днів. Для того, щоб провести розрахунок за 10 хв, необхідна ЕОМ продуктивністю 1.7 Tflops ($1.7X10^{12}$ операцій з плаваючою крапкою за секунду);

2. До категорії задач, що вимагають великого об'єму оперативної пам'яті, відносяться, наприклад, задачі гідро- і газодинаміки по розрахунку течій складної просторово-часової структури з врахуванням різних фізичних і хімічних процесів. Такі задачі є, як правило, багатвимірними, і розрахунок по кожному з напрямків хоча б для декількох точок вимагає оперативної пам'яті понад 10 Гбайт. В квантовій хімії неемпіричні розрахунки електронної структури молекул вимагають обчислювальних затрат, пропорційних N^4 чи N^5, де N умовно характеризує число молекул. Тому молекулярні системи вимужено досліджуються спрощено, через недостаток обчислювальних ресурсів;

3. Вимога забезпечення великої кількості інформації, що передається, характерна для задач гідро- і газодинаміки з змінними граничними умовами, коли обчислювальний алгоритм постійно вимагає введення нової інформації; і задач економічної оптимізації, що описують поведінку системи, яка занурена в середовище з неперервно змінними властивостями, від яких залежить стан системи;

4. Проблема обробки і зберігання великого об'єму інформації характерна для задач астрономії, спектроскопії, біології, ядерної фізики.

Засоби для проведенняя паралельних обчислень бувають:

- апаратні:
    - засоби для проведення обчислень (обчислювальна техніка);
    - обчислювальна техніка, зібрана з стандартних комплектуючих;
    - обчислювальна техніка, зібрана з спеціальних комплектуючих ;
    - засоби візуалізації;
    - засоби для зберігання і обробки даних;
- програмні:

- програмні засоби загального призначення (операційнісистеми: стандартні бібліотеки, мови програмування, компілятори, профайлери, відлагоджувачі і т.п.);
- спеціальні програмні засоби: бібліотеки (PVM, MPI); засоби об’єднання ресурсів (Dynamite, Globus і ін.).

3.3.2 Рівні розпаралелювання

Класифікація паралельності за рівнями, що відрізняються показниками абстрактності розпаралелювання задач наведена в табл.3.1. Чим "глибше" рівень, в якому наступає паралельність, тим детальнішим, малоелементнішим буде розпаралелювання, що торкається елементів програми (інструкція, елементи інструкції тощо). Чим вище розміщено рівень абстракції, тим більші блоки має паралельність.

Таблиця 3.1. Рівні позпаралелення

	Рівні	Об’єкт обробки	Приклад системи
Великоблоковий	Програмний	Робота/Задача	Мультизадачна ОС
	Процедурний	Процес	MIMD-система
	Рівень формул	Інструкція	SIMD-система
Дрібноблоковий	Біт-рівень	В межах інструкції	Машина фон Неймана

Кожний рівень має повністю різні аспекти паралельного оброблення. Методи і конструктиви даного рівня обмежуються тільки цим рівнем і не можуть бути поширені на інші рівні. Найбільший інтерес викликають рівень процедур (великоблокова, асинхронна паралельність) та рівень арифметичних виразів (малоелементна, детальна або масивна синхронна паралельність).

Програмний рівень

На цьому найвищому рівню одночасно (або щонайменше розподілено за часом) виконуються комплектні програми. Машина, що виконує ці

програми, не повинна бути паралельною ЕОМ, досить того, що в ній наявна багатозадачна операційна система (наприклад, реалізована як система розподілу часу). В цій системі кожному користувачеві відповідно до його пріоритету планувальник (Scheduler) виділяє відрізок часу різної тривалості. Користувач одержує ресурси центрального процесорного блоку тільки впродовж короткого часу, а потім стає в чергу на обслуговування.

У тому випадку, коли в ЕОМ недостатня кількість процесорів для всіх користувачів (або процесів), що, як правило, найбільш імовірно, в системі моделюється паралельне обслуговування користувачів за допомогою "квазіпаралельних" процесів.

Рівень процедур

На цьому рівні різні розділи однієї і тієї самої програми мають виконуватися паралельно. Ці розділи називаються "процесами" і відповідають приблизно послідовним процедурам. Проблеми поділяються на суттєво незалежні частини так, щоб по можливості рідше виконувати операції обміну даними між процесами, які потребують відносно великих витрат часу. В різних галузях застосування стає ясно, що цей рівень паралельності ні в якому разі не обмежується розпаралелюванням послідовних програм. існує великий ряд проблем, які потребують паралельних структур цього типу навіть тоді, коли так само, як і на програмному рівні, у користувача є тільки один процесор.

Застосування (основне) - загальне паралельне оброблення інформації, де застосовується поділ вирішуваної проблеми на паралельні задачі - частини, які вирішуються багатьма процесорами з метою підвищення обчислювальної продуктивності.

Рівень арифметичних виразів

Арифметичні вирази виконуються паралельно покомпонентно, причому в суттєво простіших синхронних методах. Якщо, наприклад, йдеться про арифметичний вираз додавання матриць, то він синхронно

розпаралелюється дуже просто тому, що кожному процесорові підпорядковується один елемент матриці.

При застосуванні n*n процесорних елементів можна одержати суму двох матриць порядку n*n за час виконання однієї операції складання (за винятком часу, потрібного на зчитування та запис даних). Цьому рівню притаманні засоби векторизації та так званої паралельності даних. Останнє поняття пов’язане з детальністю розпаралелювання, а саме - з його поширенням на оброблювані дані. Майже кожному елементу даних тут підпорядковується свій процесор, завдяки чому ті дані, що в машині фон Ноймана були пасивними. перетворюються на “активні обчислювальні пристрої”.

Рівень двійкових розрядів

На цьому рівні відбувається паралельне виконання бітових операцій в межах одного слова. Паралельність на рівні бітів можна знайти в будь-якому працюючому мікропроцесорі. Наприклад, у 8-розрядному арифметико-логічному пристрої побітова обробка виконується паралельними апаратними засобами.

3.3.2 Паралельні операції

Зовсім інший образ паралельності виникає з аналізу математичних операцій над окремими елементами даних або над групами даних. Розрізняють скалярні дані, операції над якими виконуються послідовно, і векторні дані, над якими можна виконати потрібні математичні операції паралельно. Операції, які нижче розглядаються, є основними функціями, що реалізуються у векторних та матричних ЕОМ.

Прості операції над векторами, наприклад складання двох векторів, можуть бути виконані безпосередньо синхронно і паралельно. У цьому випадку можна кожному елементу вектора підпорядкувати один процесор. При складніших операціях, таких як формування всіх часткових сум, побудова ефективного паралельного алгоритму є не зовсім очевидною.

Основні поняття паралелізму

Розглянемо алгоритми розпаралелення типових задач незалежно від конкретної програмної і платформенної реалізації (рис 3.2-3.4). Розпаралелити задачу можна не єдиним способом. Алгоритми розпаралелення зручно графічно зобразити в вигляді розгалужених дерев.

Перший етап: Розбиття задачі на незалежні підзадачі.

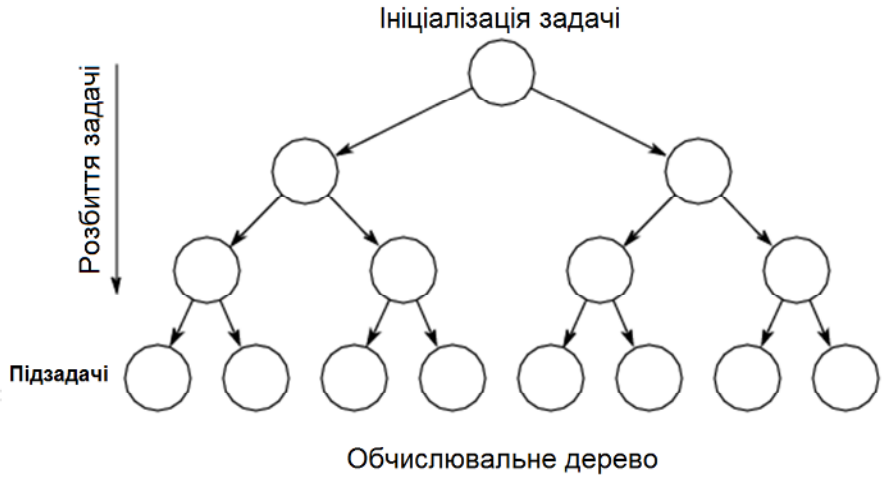


Рис. 3.2 Розбиття задачі на незалежні підзадачі.

Другий етап: Призначення конкретних процесорів для виконання кожної підзадачі.

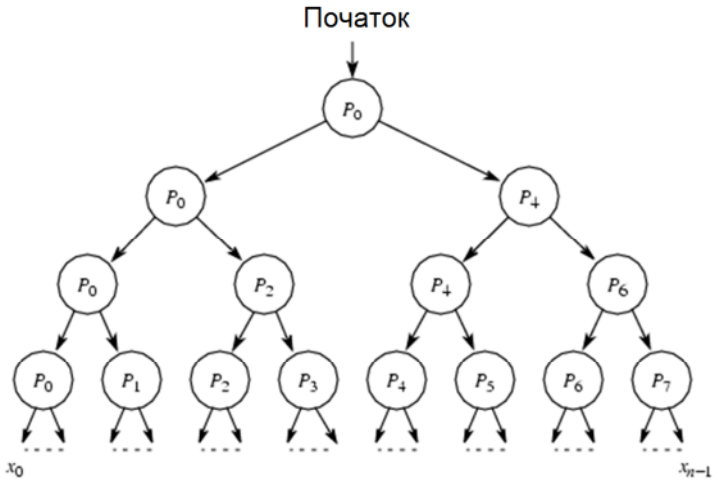


Рис. 3.3 Розподіл підзадач між процеосрами

Третій етап: Збирання результатів роботи окремих процесорів.

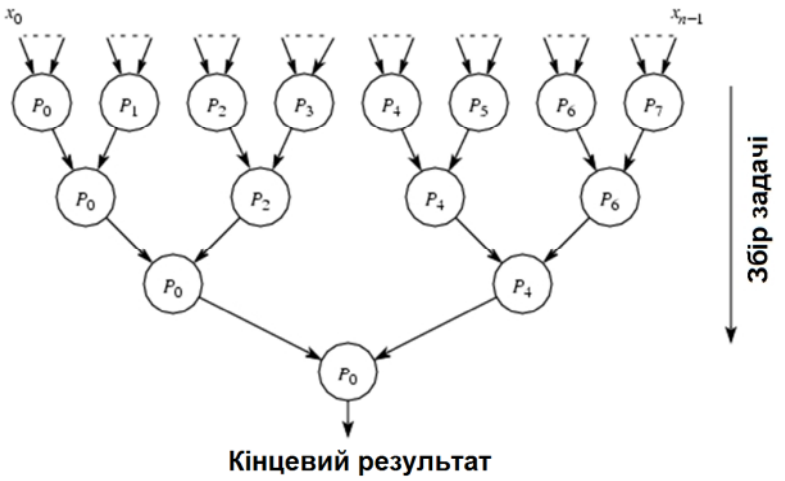


Рис. 3.4 Збирання результатів підзадач в головному процесорі.

Контрольні питання до розділу 3

1. Що входить до складу операційної системи?
2. Етапи завантаження операційної системи.
3. Функції операційних систем
4. Види операційних систем
5. Що таке процес і потік?
6. В чому суть взаємодії процесів?
7. Назвіть базові механізми міжпроцесової взаємодії
8. Які є примітиви передавання повідомлень?
9. Назвіть основні складові технології передавання повідомлень?
10. Що таке паралелізм?
11. Що таке розподілені обчислення?
12. Назвіть приклади задач для розподілених обчислень
13. Назвіть і проаналізуйте рівні розпаралелення процесів
14. Назвіть основні етапи розпаралелення задач?

РОЗДІЛ 4. ОРГАНІЗАЦІЯ ПАМ'ЯТІ ТА ВВЕДЕННЯ-ВИВЕДЕННЯ В КОМП'ЮТЕРНИХ СИСТЕМАХ

Тема 4.1. Системи введення-виведення (СВВ)

4.1.1 Функціонування комп'ютерного введення-виведення

Набір функцій СВВ практично не залежить від типу ЕОМ, проте їх конкретне розподіл між різними апаратними та програмними компонентами СВВ значною мірою визначається призначенням ЕОМ, умовами та режимами використання, її архітектурою, характеристиками продуктивності, вартості і т.д. Це розподіл функцій при виконанні операцій введення-виведення називається структурною організацією СВВ.

Сучасні ЕОМ будуються як системи з змінним складом устаткування, що дозволяє на одній і тій же машині вирішувати, хоча і з різним ступенем ефективності, завдання різних класів. Система із змінним складом устаткування будується з деякого набору пристроїв-модулів, що включає ЦП, модулі ОЗП; різні компоненти-модулі СВВ. Зміна конфігурації ЕОМ забезпечується за рахунок кабельних з'єднань і модифікації програм управління апаратними модулями. Концепція змінного складу обладнання вимагає стандартизації форматів повідомлень, алгоритмів керування обміном між апаратними модулями, способів додавання нових програмних модулів. Така стандартизація визначає архітектуру системи (сімейства або ряду) ЕОМ, яка допускає в певних межах зміна кількості і складу модулів (в першу чергу периферійних пристроїв (ПП)) і забезпечує при цьому інформаційну та програмну сумісність. Під архітектурою обчислювальної системи при цьому розуміють загальну логічну організацію цифрової обчислювальної системи, визначальну процес обробки даних, методи кодування, склад, призначення, принципи взаємодії технічних засобів і програмного забезпечення). Інформаційна сумісність всіх ЕОМ однієї системи досягається єдиними способами кодування інформації та єдиним форматом даних. Програмна

сумісність означає можливість виконання програм (без будь-яких змін) у різних конфігураціях ЕОМ, що досягається єдиною системою команд і однаковою організацією операційних систем. Апаратна сумісність модулів забезпечується уніфікованою системою сполучення - інтерфейсами різних рівнів - і єдиними способами управління.

Реалізація ЕОМ у вигляді системи із змінним складом устаткування дозволяє розширити номенклатуру ПП і спростити спілкування користувача з машиною при вирішенні наукових завдань за рахунок розширення алфавіту повідомлень і представлення результатів у вигляді таблиць з коментарями та графіків; при вирішенні завдань управління стає можливим відображати і документувати хід процесу управління.

Система ЕОМ об’єднує різні моделі, орієнтовані на вирішення переважно одного класу задач, хоча ці моделі і розрізняються по продуктивності. У моделях ЕОМ різних систем можуть використовуватися ПП одного функціонального призначення, але істотно відрізняються за своїми характеристиками. З точки зору орієнтації машин на той чи інший клас завдань, організації СВВ, складу і характеристик основних ПУ можна виділити наступні класи ЕОМ.

Персональні ЕОМ (ПЕОМ) призначені для роботи з одним користувачем в режимі індивідуального доступу. Цей клас охоплює широке коло ПЕОМ від професійних до домашнього користування. СВВ персональних ЕОМ відрізняються порівняно простотою організації, наявністю обмеженого числа дешевих ПП, серед яких найбільш поширені клавіатури і маніпулятори для введення інформації, дешеві пристрої відображення, жорсткі магнітні диски типу HDD, а також послідовні пристрої друку. Більшість сучасних ПЕОМ обладнуються засобами підключення до локальних мереж; вони орієнтовані на непрофесійного користувача, тому одне з основних вимог до СВВ полягає в організації найбільш природного спілкування - у формі діалогу, за допомогою графіки, а в майбутньому - за допомогою й мови.

Управляючі мікроЕОМ орієнтовані на роботу в реальному масштабі часу з управління об’єктами і технологічними процесами. СВВ керуючих мікроЕОМ повинна забезпечувати швидку реакцію на зміни в стані керованих об’єктів. Характерними ПП для ЕОМ цього класу є цифроаналогові (ЦАП) і аналого-цифрові (АЦП) перетворювачі і ПВВ дискретних сигналів, що входять до складу пристроїв узгодження з об’єктом (ПУО) ; крім того, керуючі мікроЕОМ часто об’єднують з іншими ЕОМ в багатомашинні обчислювальні комплекси та системи за допомогою різних пристроїв сполучення, включаючи локальні мережі. Конструктивно мікроЕОМ часто виконують у вигляді модулів, вбудованих в системи управління.

Міні ЕОМ спочатку призначалися для управління складними об’єктами і технологічними процесами, однак зі зростанням обчислювальних можливостей основними областями використання міні ЕОМ стали багато користувачів системи для автоматизації проектно-конструкторських робіт, розробки програмного забезпечення МП систем, СВВ міні ЕОМ зберегли риси керуючих машин, проте склад ПП значно розширився. Міні ЕОМ могли працювати як в режимі реального часу, при цьому в якості основних ПП використовувалися ПУО, так і в діалоговому режимі колективного доступу, при цьому найбільш поширеними ПП були дисплеї, ЗП великої ємності, а також пристрої сполучення з каналами зв’язку для організації дистанційного доступу. Найбільш характерними прикладами міні ЕОМ були машини сімейств СМ (наприклад, СМ-2М), СМ-4 (СМ-1420) і СМ-1700.

ЕОМ загального призначення призначалися для вирішення широкого кола завдань наукового, інженерного та економічного характеру. На базі ЕОМ цього класу створювалися обчислювальні центри колективного користування. СВВ в ЕОМ загального призначення були орієнтовані на роботу в пакетному режимі і в режимі колективного доступу; до складу СВВ входили різноманітні складні ПП високої продуктивності і пристрої сполучення з каналами зв’язку. При проектуванні СВВ цього класу машин

одним з основних вимог було забезпечення збалансованості СВВ і засобів обробки. Під збалансованістю розуміють таке співвідношення між продуктивністю ЦП і пропускною здатністю СВВ, при якому обсяг переробляється за одиницю часу інформації відповідає обсягу введеної та виведеної інформації за той же інтервал часу. За умови повної збалансованості ЦП і СВВ працюють без простоїв, чим досягається найкраще ставлення продуктивності до вартості обладнання.

СуперЕОМ по суті являють собою високопродуктивні проблемно-орієнтовані комп’ютери, призначені для виконання досить вузького класу задач. Залежно від класу вирішуваних завдань суперЕОМ, швидкодія яких в даний час досягає сотень і тисяч мільярдів операцій в секунду, можуть комплектуватися різними, часом дуже складними і дорогими ПП. При створенні суперЕОМ основним завданням є досягнення максимальної швидкодії. Щоб розвантажити центральну частину суперЕОМ від участі у виконанні операцій введення-виведення, часто організацію та управління цими операціями покладають на додаткові ЕОМ.

4.1.2 Функції та структури систем введення-виведення.

Завдання СВВ (її внутрішньої частини) полягає в організації та управлінні процесом передачі інформації від ПУ в ОП машини при введенні і у зворотному напрямку при виведенні, тобто у виконанні операцій введення-виведення.

Основні функції СВВ і способи їх реалізації. З точки зору СВВ будь ПП являє собою генератор даних (ГенД) квантів даних $D_{ij,}$ який може запускатися в роботу сигналами C_{ij} від керуючих компонентів СВВ і повідомляти їм про свій стан сигналами $S_{ij,}$ як показано на рис 4.1.

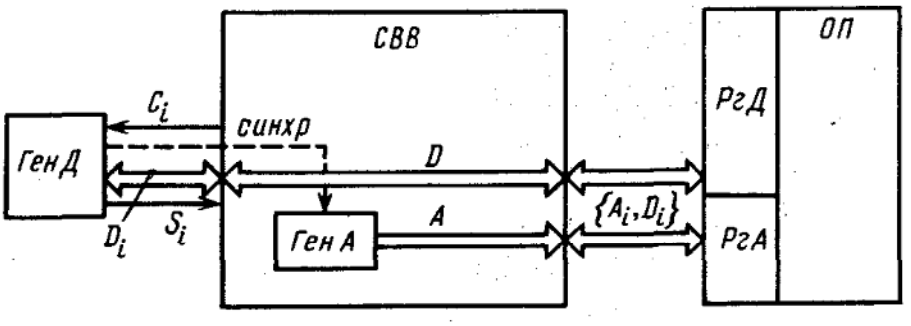


Рис 4.1 Взаємодія СВВ з пам’яттю.

Тривалість інтервалів формування послідовних квантів інформації в такому генераторі і кванти даних істотно відрізняються від тривалості інтервалів обробки і квантів в центральних пристроях машини. Тому основні функції СВВ можна сформулювати наступним чином:

- Перетворення квантів (або форматів) інформації, прийнятих від ПП при введенні, у формати ЦП і ОП; зворотне перетворення - при виведенні;
- Визначення місця в ОП, де повинен бути розміщений сформований машинний квант при введенні або звідки повинен бути вибраний при виведенні, тобто формування поточного адреси ОП. Таким чином, кошти СВВ можуть розглядатися як генератор адрес гена оперативної пам’яті $A_{ij,}$ сформованих синхронно для кожного генерується в ПП кванта даних $D_{ij.}$ ГенД і лінії пов’язані з регістрами даних РгД та адреси РгА в ОП машини;
- Формування керуючих сигналів для роботи ПП в різних режимах, завдання типу виконуваної операції в ПП і т.д.;
- Отримання та обробка сигналів $S_{ij,}$ що характеризують стан ПУ, моживість виконання ним тих чи інших дій;
- Отримання наказів від центральних пристроїв на виконання операцій ввеення-виведення, формування повідомлень про стан СВВ;
- Синхронізація процесів у ЦУ і ПУ, узгодження швидкостей їх роботи.

Найпростіша реалізація перерахованих функцій можлива при центрально-синхронному принципі управління. При цьому синхронізація

всіх пристроїв ЕОМ здійснюється від єдиного центрального пристрою керування (ЦПК), а всі передачі даних від ПП або до нього виробляються через АЛП. Структура ЕОМ з центрально-синхронним принципом управління показана на рис. 4.2. Центральна частина машини заштрихована, суцільними лініями показані зв'язку для передачі даних і адрес; штриховими - зв'язку для керуючих сигналів і команд.

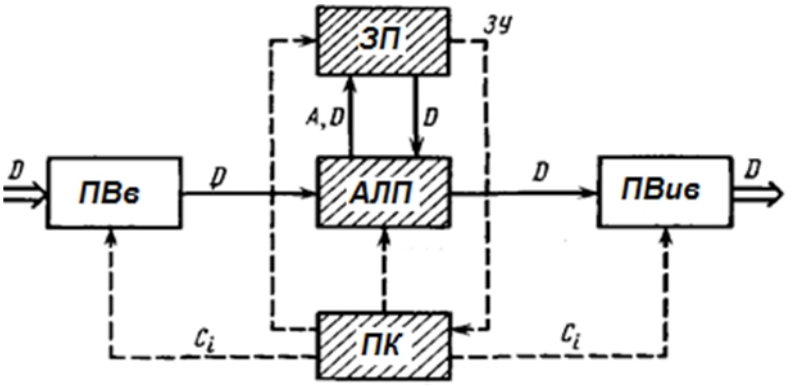


Рис 4.2 Централізовано-синхронне управління

При централізовано-синхронному управлінні всі операції обробки та введення-виведення повинні виконуватися послідовно. У системі команд машини повинні бути передбачені спеціальні команди операцій введення-виведення. Одна виконувана команда служить для передачі одного кванта інформації; при необхідності передачі масиву інформації повинні бути організовані циклічні програми. Оскільки в операціях введення-виведення беруть участь пристрої з істотно різною швидкодією, то тривалість одиничної операції ТВВ визначається швидкодією самого повільного пристрою (тобто ПП) і суттєво перевищує тривалість операції обробки T_a. Тривалість рішення задачі для такої ЕОМ безпосередньо залежить від швидкодії використовуваних ПУ і частки операцій введення-виведення. Централізовано-синхронний принцип управління вимагає менших апаратних витрат, він був характерний для перших ЕОМ, що використовуються в наукових розрахунках. Збільшення частки операцій введення-виведення при переході до завдань обробки даних робило цей принцип практично

непридатним. Крім того, при використанні цього принципу неможливо побудова систем із змінним складом устаткування.

Поліпшити продуктивність ЕОМ можна за рахунок організації паралельного виконання операцій обробки та введення-виведення. При наявності коштів автономного управління роботою ПП безпосередню участь центральних пристроїв в обміні інформацією з ПК може обмежуватися тактом передачі ПК, який, як зазначалося вище, значно коротше такту підготовки. Крім того, при операціях обробки лише частину часу виконання команди в АЛП пішов на звернення до ОЗП, тому виділення спеціальних засобів управління і доступу до пам’яті з боку ПУ дозволяє істотно підвищити продуктивність ЕОМ.

Однак при цьому має бути реалізований асинхронний принцип управління, що забезпечує незалежність роботи ПК, ОЗП і АЛП. ПК починає роботу по команді запуску, після чого працює автономно, готуючи квант інформації. ЦП продовжує виконання поточної програми. Підготувавши квант інформації, ПУ посилає сигнал запиту процесору і ЦП призупиняє виконання поточної програми для отримання підготовленого кванта. По закінченні роботи ПУ посилає процесору сигнал, підтверджує завершення операції. Паралельна робота ПУ і ЦП здійснюється в тактах підготовки кванта інформації в ПУ. Під час тактів передачі центральні та периферійні пристрої використовуються спільно, тому в ці моменти для організації обміну необхідно синхронізувати їх роботу. З цією метою використовуються переривання та призупинення, розглянуті нижче. При паралельному виконанні операцій обробки та введення-виведення тривалість рішення задачі тим менше, чим вище коефіцієнт (Кп) перекриття, або збігу в часі, операцій обробки та введення-виведення, що характеризує, яку частку циклу ПУ процесор і ПК можуть працювати незалежно. За відсутності перекриття, тобто при послідовному виконанні операцій, Кп = 0.

У реальних системах залежно від ступеня і способу реалізації умов, необхідних для паралельної роботи центральних і периферійних пристроїв,

цей коефіцієнт може приймати будь-які значення в інтервалі (0,1). Для збільшення Кп необхідно виконати наступні умови:

-Управління ПК при підготовці квантів інформації має здійснюватися автономними схемами, що працюють незалежно від ЦП;

-В ЕОМ мають бути передбачені засоби зв'язку для передачі квантів інформації між ПК і ОП, минаючи АЛУ, так звані засоби прямого доступу до пам'яті;

-Повинні бути передбачені засоби для синхронізації паралельного виконання асинхронних процесів обробки в центральних пристроях і підготовки квантів інформації в ПП;

-Протягом всього процесу введення-виведення ЦП повинен бути настільки завантажений операціями обробки, щоб причиною простоїв, що виникають в них, не була нестача вихідних даних або команд.

Структура ЕОМ з асинхронним паралельним виконанням операцій обробки та введення-виведення показана на рис. 4.3. У цій структурі передбачені додаткові тракти передачі даних між СВВ, Пвив та ЗП, тим самим обмін відбувається, минаючи АЛП. Управління роботою ПП, формування поточних адрес і запитів до пам'яті здійснюється за допомогою спеціальних схем управління (каналу введення-виведення - КВВ), взаємодія, яких з ЦП реалізується через систему переривань і призупинок.

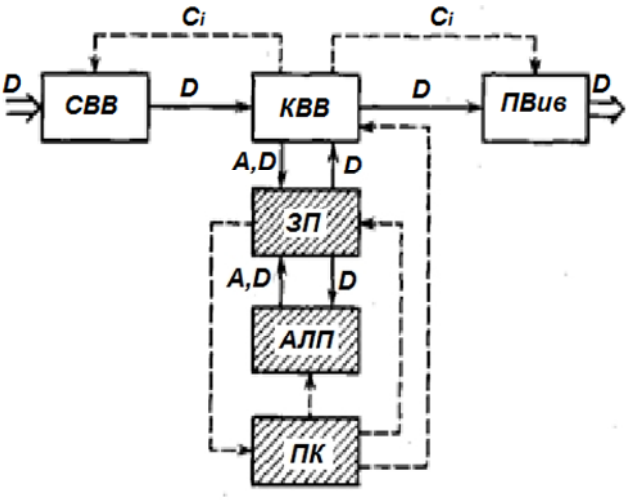


Рис. 4.3 Структура ЕОМ з асинхронним паралельних виконанням операцій

4.1.3 Засоби суміщення операцій обробки та введення-виведення.

Основними засобами, що дозволяють поєднати операції обробки та введення-виведення, є переривання і призупинення. Ці засоби забезпечують можливість взаємодії асинхронно протікають процесів. Окрім засобів переривання і припинень для паралельного виконання операції широко використовуються різні види буферизації.

Переривання - процес перемикання ЦП з однієї програми на іншу за зовнішнім сигналом із збереженням інформації для подальшого відновлення перерваної програми. Необхідність у перериванні виникає в тому випадку, якщо деяка зовнішня по відношенню до ЦП подія вимагає від нього негайної реакції.

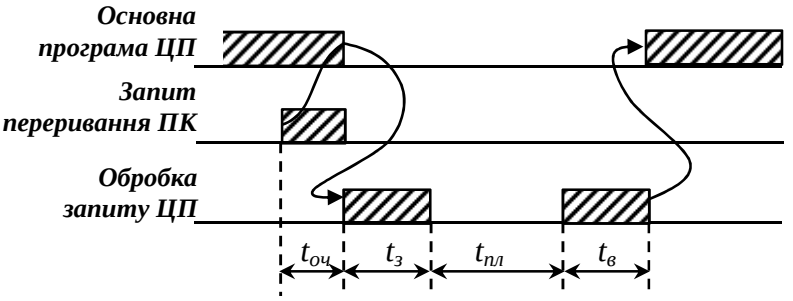


Рис. 4.4 Процес переривання обчислень

ПК при виникненні події, що вимагає реакції з боку ЦП, формує сигнал, який називається запитом переривання. Він може надходити в ЦП в довільні моменти часу асинхронно по відношенню до виконання програми, тому запити переривань запам’ятовуються на спеціальному регістрі, що називається регістром запитів переривань (РгЗП).

Стан РгЗП аналізується апаратними або програмними засобами в певні моменти виконання програми або команди. У простому випадку після виконання кожної команди схеми управління проводять опитування стану РгЗП і за наявності в ньому одиниці переходять до виконання переривання. Інтервал часу очікування між моментом надходження сигналу запиту

переривання в РгЗП і моментом початку обробки переривання називають часом реакції на переривання. Обробка переривання включає в себе етапи запам’ятовування стану перерваної програми і переходу до виконання програми, яка перериває; власне виконання програми, яка перериває; відновлення стану перерваної програми і повернення до її виконання .

Призупинення - процес, при якому засоби управління, що працюють автономно від ЦП, затримують його роботу на час циклу пам’яті, при цьому ОЗП безпосередньо зайнято прийманням або видачею інформації для іншого пристрою. Під час призупинень поточний стан процесора не змінюється, але виконання команди затримується до звільнення ОЗП. Призупинення забезпечують високий ступінь суміщення операцій обробки та введення-виведення. Суміщення має тим вищий ступінь, чим менше тривалість циклу пам’яті щодо тривалості команди процесора.

Тема 4.2. Організація пам’яті в ПРКС

4.2.1 Загальні питання організація пам’яті.

У обчислювальних системах, об’єднуючих множину паралельно працюючих процесорів або машин, задача правильної організації пам’яті є однією з найважливіших. Різниця між швидкодією процесора і пам’яті завжди була каменем спотикання в однопроцесорних ОС. Багатопроцесорність ОС приводить ще до однієї проблеми — проблеми одночасного доступу до пам’яті з боку декількох процесорів.

Залежно від того, яким чином організована пам’ять багатопроцесорних (багатомашинних) систем, розрізняють обчислювальні системи із загальною пам’яттю (shared memory) і ОС з розподіленою пам’яттю (distributed memory). У системах із загальною пам’яттю (її часто називають також спільно використовуваною або пам’яттю, що розділяється) пам’ять ОС розглядається як загальний ресурс, і кожен з процесорів має повний доступ до всього адресного простору. Системи із загальною пам’яттю називають сильно

зв’язаними (closely coupled systems). Подібна побудова обчислювальних систем має місце як в класі SIMD, так і в класі MIMD. Іноді, щоб підкреслити цю обставину, вводять спеціальні підкласи, використовуючи для їх позначення абревіатури SM-SIMD (Shared Memory SIMD) і SM-MIMD (Shared Memory MIMD).

У варіанті з розподіленою пам’яттю кожному з процесорів додається власна пам’ять. Процесори об’єднуються в мережу і можуть при необхідності обмінюватися даними, що зберігаються в їх пам’яті, передаючи один одному так звані повідомлення. Такий вид ОС називають слабо зв’язаними (loosely coupled systems). Слабо зв’язані системи також зустрічаються як в класі SIMD, так і в класі MIMD, і інший раз, щоб підкреслити дану особливість, вводять підкласи DM-SIMD (Distributed Memory SIMD) і DM-MIMD (Distributed Memory MIMD).

В деяких випадках обчислювальні системи із загальною пам’яттю називають мультипроцесорами, а системи з розподіленою пам’яттю — мультикомпъютерами.

Відмінність між загальною і розподіленою пам’яттю — це різниця в структурі віртуальної пам’яті, тобто в тому, як пам’ять виглядає з боку процесора. Фізично майже кожна система пам’яті розділена на автономні компоненти, доступ до яких може проводитися незалежно. Загальну пам’ять від розподіленої відрізняє те, яким чином підсистема пам’яті інтерпретує надіслану процесором адресу комірки пам’яті. Для прикладу вважатимемо, що процесор виконує команду load R0, I, що означає «Завантажити регістр R0 вмістом комірки I». У разі загальної пам’яті I — це глобальна адреса, і для будь-якого процесора указує на одну і ту ж комірку. У розподіленій системі пам’яті I — це локальна адреса. Якщо два процесори виконують команду load R0, I, то кожен з них звертається до I-ої комірки в своїй локальній пам’яті, тобто до різних комірок, і в регістри R0 можуть бути завантажені неоднакові значення.

Пам’ять з чергуванням адрес.

Фізично пам’ять обчислювальної системи складається з декількох модулів (банків), при цьому істотним питанням є те, як в цьому випадку розподілений адресний простір (набір всіх адрес, які може сформувати процесор). Один із способів розподілу віртуальних адрес по модулях пам’яті полягає в розбитті адресного простору на послідовні блоки. Якщо пам’ять складається з *n* банків, то комірка з адресою *i* при поблочному розбитті знаходитиметься в банку з номером *i/n*. У системі пам’яті з чергуванням адрес (interleaved memory) послідовні адреси розташовуються в різних банках: комірка з адресою *i* знаходиться в банку з номером *i* mod *n*. Нехай, наприклад, пам’ять складається з чотирьох банків, по 256 байт в кожному. У схемі, орієнтованій на блокову адресацію, першому банку будуть виділені віртуальні адреси 0-255, другому — 256-511 і так далі. В схемі з чергуванням адрес послідовні комірки в першому банку матимуть віртуальні адреси 0, 4, 8 .., у другому банку — 1, 5, 9 і так далі (рис. 4.5,*а*).

Розподіл адресного простору по модулях дає можливість одночасної обробки запитів на доступ до пам’яті, якщо відповідні адреси відносяться до різних банків . Процесор може в одному з циклів зажадати доступ до комірки *i*, а в наступному циклі — до комірки *j*. Якщо *i* і *j* знаходяться в різних банках, інформація буде передана в послідовних циклах. Тут під циклом розуміється цикл процесора, тоді як повний цикл пам’яті займає декілька циклів процесора. Таким чином, в даному випадку процесор не повинен чекати, поки буде завершений повний цикл звернення до комірки *i*. Розглянутий прийом дозволяє підвищити пропускну спроможність: якщо система пам’яті складається з достатнього числа банків, є можливість обміну інформацією між процесором і пам’яттю з швидкістю одне слово за цикл процесора, незалежно від тривалості циклу пам’яті.

Системи пам’яті часто забезпечують додаткову гнучкість при зчитуванні елементів векторів. У деяких системах можливе одночасне завантаження кожного *n*-го елементу вектора, наприклад, при зчитуванні

елементів вектора *v*, що зберігається в послідовних елементах пам’яті при *n* = 4, пам’ять поверне v_1, v_4, v_8. Інтервал між елементами називають кроком по індексу або «страйдом» (stride). Одним з цікавих застосувань цієї властивості може служити доступ до матриць. Якщо крок по індексу на одиницю більше числа рядків в матриці, одиночний запит на доступ до пам’яті поверне всі діагональні елементи матриці (рис. 4.5, *б*). Відповідальність за те, щоб всі зчитані елементи матриці розташовувалися в різних банках, лягає на програміста.

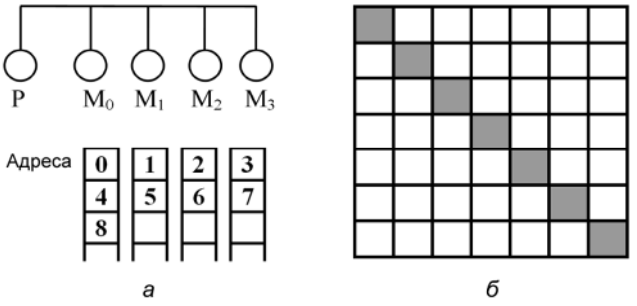


Рис. 4.5 Пам’ять з чергуванням адрес: *а* — розподіл адрес
б — елементи, витягувані з кроком 9 з масиву 8 х 8.

4.2.2 Моделі архітектури пам’яті обчислювальних систем

Як для спільно використовуваної пам’яті так і для розподіленої пам’яті реалізується декілька моделей архітектур систем пам’яті.

На рис. 4.6 наведена класифікація таких моделей пам’яті в обчислювальних системах класу MIMD (аналогічно і для класу SIMD).

У системах із загальною пам’яттю всі процесори мають рівні можливості по доступу до єдиного адресного простору. Єдина пам’ять може бути побудована як одноблочна або за модульним принципом, але зазвичай практикується другий варіант.

Обчислювальні системи із загальною пам’яттю, де доступ будь-якого процесора до пам’яті проводиться одинаково і займає однаковий час, називають системами з однорідним доступом до пам’яті і позначають абревіатурою UMA (Uniform Memory Access). Це найбільш поширена

архітектура пам’яті паралельних ОС із загальною пам’яттю.

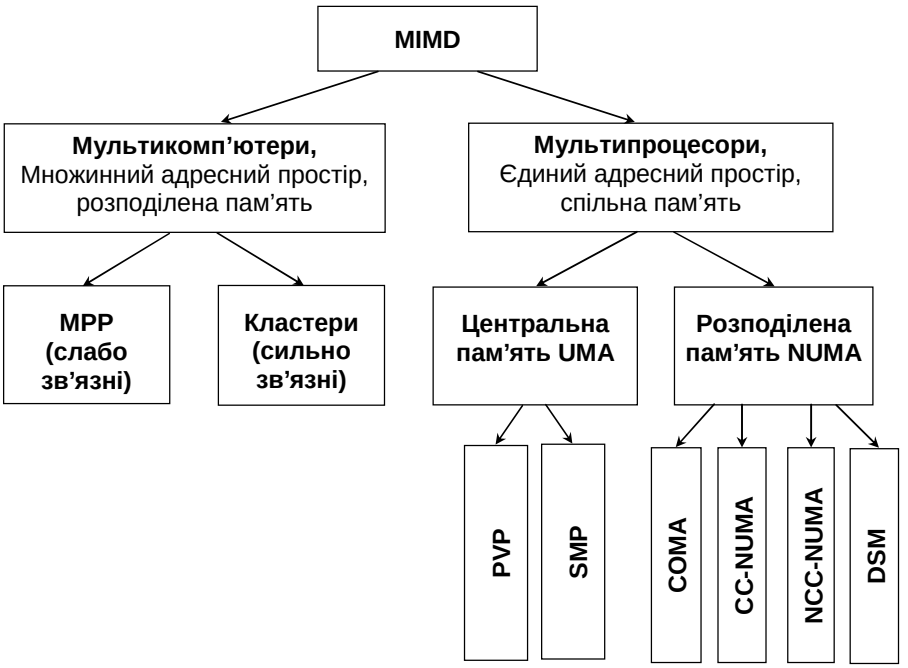


Рис. 4.6. Класифікація моделей архітектури пам’яті обчислювальних систем

Технічно UMA-системи припускають наявність вузла, що з’єднує кожен з *n* процесорів з кожним з *m* модулів пам’яті. Простий шлях побудови таких ОС — об’єднання декількох процесорів (P_i) з єдиною пам’яттю *(Mp)* за допомогою загальної шини, що показано на рис. 4.7, *а*. В цьому випадку, проте, в кожен момент часу обмін по шині може вести тільки один з процесорів, тобто процесори повинні змагатися за доступ до шини. Коли процесор *P*, вибирає мул пам’яті команду, решта процесорів *Pj (i S j)* повинна чекати, поки шина звільниться. Якщо в систему входять тільки два процесори, вони в змозі працювати з продуктивністю, близькою до максимальної, оскільки їх доступ до шини можна чергувати: поки один процесор декодує і виконує команду, інший має право використовувати шину для вибірки з пам’яті наступної команди. Проте коли додається третій процесор, продуктивність починає падати. За наявності на шині десяти процесорів крива швидкодії шини (рис. 4.7, в) стає горизонтальною, так що додавання 11-го процесора вже не дає підвищення продуктивності. Нижня крива на цьому малюнку ілюструє той факт, що пам’ять і шина володіють фіксованою пропускною спроможністю, визначуваною комбінацією

тривалості циклу пам’яті і протоколом шини, і в багатопроцесорній системі із загальною шиною ця пропускна спроможність розподілена між декількома процесорами. Якщо тривалість циклу процесора більше в порівнянні з циклом пам’яті, до шини можна підключати багато процесорів. Проте фактично процесор зазвичай набагато швидше за пам’ять, тому дана схема широкого застосування не знаходить.

Альтернативний спосіб побудови багатопроцесорної ОС із загальною пам’яттю на основі UMA показаний на рис. 4.7, *г*. Тут шина замінена комутатором, що маршрутизує запити процесора до одного з декількох модулів пам’яті. Не дивлячись на те що є декілька модулів пам’яті, всі вони входять в єдиний віртуальний адресний простір. Перевага такого підходу в тому, що комутатор може обслуговувати паралельно декілька запитів. Кожен процесор може бути з’єднаний зі своїм модулем пам’яті і мати доступ до нього на максимально допустимій швидкості. Суперництво між процесорами може виникнути при спробі одночасного доступу до одного і того ж модуля пам’яті. В цьому випадку доступ дістає тільки один процесор, а інші — блокуються.

Рис. 4.7. Загальна пам’ять: *а* – об’єднання процесорів за допомогою шини, *б* – система з локальними кешами; *в* – продуктивність системи як функція від числа процесорів на шині; *г* – багатопроцесорна ОС із загальною пам’яттю, що складається з окремих модулів.

На жаль, архітектура UMA не дуже добре масштабується. Найбільш поширені системи містять 4-8 процесорів, значно рідше до 32-64 процесорів.

Крім того, подібні системи не можна віднести до відмовостійких, оскільки відмова одного процесора або модуля пам’яті приводить до відмови всієї ОС.

Іншим підходом до побудови ОС із загальною пам’яттю є неоднорідний доступ до пам’яті, що позначається як NUMA (Non-Uniform Memory Access). Тут як і раніше фігурує єдиний адресний простір, але кожен процесор має локальну пам’ять. Доступ процесора до власної локальної пам’яті проводиться безпосередньо, що набагато швидше, ніж доступ до віддаленої пам’яті через комутатор або мережу. Така система може бути доповнена глобальною пам’яттю, тоді локальні запам’ятовуючі пристрої, грають роль швидкої кеш-пам’яті для глобальної пам’яті. Подібна схема може поліпшити продуктивність ОС, але не в змозі необмежено відстрочити вирівнювання прямої продуктивності. За наявності у кожного процесора локальної кеш-пам’яті (рис. 4.3, *б*) існує висока вірогідність *(p* > 0,9) того, що потрібні команда або дані вже знаходяться в локальній пам’яті. Висока вірогідність попадання в локальну пам’ять істотно зменшує число звернень процесора до глобальної пам’яті і, таким чином, веде до підвищення ефективності. Місце зламу кривої продуктивності, що відповідає точці, в якій додавання процесорів ще залишається ефективним (верхня крива на рис. 4.3, *в*), тепер переміщується в область 20 процесорів, а точка, де крива стає горизонтальною, — в область 30 процесорів.

В рамках концепції NUMA реалізується декілька різних підходів, що позначаються абревіатурами COMA, CC-NUMA і NCC-NUMA.

У архітектурі тільки з кеш-пам’яттю (COMA, Cache Only Memory Architecture) локальна пам’ять кожного процесора побудована як велика кеш-пам’ять для швидкого доступу з боку «свого» процесора. Кеші всіх процесорів в сукупності розглядаються як глобальна пам’ять системи. Власне глобальна пам’ять відсутня. Принципова особливість концепції COMA виражається в динаміці. Тут дані не прив’язані статично до певного модуля пам’яті і не мають унікальної адреси, що залишається незмінним протягом всього часу існування змінної. У архітектурі COMA дані переносяться в кеш-

пам’ять того процесора, який останнім їх запитав, при цьому змінна не фіксована унікальною адресою і в кожен момент часу може розміщуватися в будь-якій фізичній комірці. Перенесення даних з одного локального кеша в інший не вимагає участі в цьому процесі операційної системи, але має на увазі складну і дорогу апаратуру управління пам’яттю. Для організації такого режиму використовують так звані каталоги кешів. Відзначимо також, що остання копія елементу даних ніколи з кеш-пам’яті не видаляється.

Оскільки в архітектурі COMA дані переміщаються в локальну кеш-пам’ять процесора-власника, такі ОС в плані продуктивності володіють істотною перевагою над іншою архітектурою NUMA. З іншого боку, якщо єдина змінна або дві різні змінні, що зберігаються в одній комірці одного і того ж кеша, потрібні двом процесорам, ця комірка кеша повинна переміщатися між процесорами туди і назад при кожному доступі до даних. Такі ефекти можуть залежати від деталей розподілу пам’яті і приводити до непередбачуваних ситуацій.

Модель кеш-когерентного доступу до неоднорідної пам’яті (CC-NUMA, Cache Coherent Non-Uniform Memory Architecture) принципово відрізняється від моделі COMA. У системі CC-NUMA використовується не кеш-пам’ять, а звичайна фізично розподілена пам’ять. Не відбувається ніякого копіювання сторінок або даних між елементами пам’яті. Немає ніякої програмно реалізованої передачі повідомлень. Існує просто одна карта пам’яті, з частинами які фізично зв’язані шинами даних, і «розумні» апаратні засоби. Апаратна реалізована кеш-когерентність означає, що не потрібно додаткового програмного забезпечення для збереження безлічі копій оновлених даних або їх передачі. Зі всім цим справляється апаратний рівень. Доступ до локальних модулів пам’яті в різних вузлах системи може проводитися одночасно і відбувається швидше, ніж до віддалених модулів пам’яті.

Відмінність моделі з кеш-некогерентним доступом до неоднорідної пам’яті (NCC-NUMA, Non-Cache Coherent Non-Uniform Memory Architecture)

від CC-NUMA очевидно з назви. Архітектура пам’яті припускає єдиний адресний простір, але не забезпечує узгодженості глобальних даних на апаратному рівні. Управління використанням таких даних повністю покладається на програмне забезпечення (застосування або компілятори). Не дивлячись на цю обставину, яка фактично є недоліком архітектури, вона виявляється вельми корисною при підвищенні продуктивності обчислювальних систем з архітектурою пам’яті типу DSM.

В цілому, ОС із загальною пам’яттю, побудовані по схемі NUMA, називають архітектурою з віртуальною загальною пам’яттю (virtual shared memory architectures). Даний вид архітектури, зокрема CC-NUMA, останнім часом розглядається як самостійний і досить перспективний вид обчислювальних систем класу MIMD, тому такі ОС нижче будуть розглянуті детальніше.

4.2.3 Моделі архітектур з розподіленою пам’яттю

У системі з розподіленою пам’яттю кожен процесор володіє власною пам’яттю і здатний звертатися тільки до неї. Деякі автори називають цей тип систем багатомашинними ОС або мулътикомп’ютерами, підкреслюючи той факт, що блоки, з яких будується система, самі по собі є невеликими обчислювальними системами з процесором і пам’яттю. Моделі архітектури з розподіленою пам’яттю прийнято позначати як архітектура без прямого доступу до віддаленої пам’яті (NORMA, No Remote Memory Access). Така назва виходить з того факту, що кожен процесор має доступ тільки до своєї локальної пам’яті. Доступ до віддаленої пам’яті (локальної пам’яті іншого процесора) можливий тільки шляхом обміну повідомленнями з процесором, якому належить пам’ять, що адресується.

Подібна організація характеризується рядом переваг. По-перше, при доступі до даних не виникає конкуренція за шину або комутатори — кожен процесор може повністю використовувати смугу пропускання тракту зв’язку з власною локальною пам’яттю. По-друге, відсутність загальної шини

означає, що немає і пов’язаних з цим обмежень на число процесорів: розмір системи обмежує тільки мережа, об’єднуюча процесори. По-третє, знімається проблема когерентності кеш-пам’яті. Кожен процесор має право самостійно міняти свої дані, не піклуючись про узгодження копій даних у власній локальній кеш-пам’яті з кешами інших процесорів.

Основний недолік ОС з розподіленою пам’яттю полягає в складності обміну інформацією між процесорами. Якщо якийсь з процесорів потребує дані з пам’яті іншого процесора, він повинен обмінятися з цим процесором повідомленнями. Це приводить до двох видів витрат:

- потрібний час для того, щоб сформувати і переслати повідомлення від одного процесора до іншого;
- для забезпечення реакції на повідомлення від інших процесорів приймаючий процесор повинен отримати запит переривання і виконати процедуру обробки цього переривання.

Структура системи з розподіленою пам’яттю приведена на рис. 4.4. У лівій частині (рис 4.8, *а*) показаний один процесорний елемент (ПЕ). Він включає власне процесор (Р), локальну пам’ять (М) і два контроллери вводу/виводу ($К_0$ і $К_1$). У правій частині (рис 4.8, *б*) показана чотирипроцесорна система, що ілюструє, яким чином повідомлення пересилаються від одного процесора до іншого. По відношенню до кожного ПЕ решту всіх процесорних елементів можна розглядати просто як пристрої вводу/виводу. Для посилки повідомлення в іншій ПЕ процесор формує блок даних в своїй локальній пам’яті і сповіщає свій локальний контроллер про необхідність передачі інформації на зовнішній пристрій. По мережі міжз’єднань це повідомлення пересилається на приймальний контроллер вводу/виводу приймаючого ПЕ. Останній знаходить місце для повідомлення у власній локальній пам’яті і повідомляє процесор-джерело про отримання повідомлення.

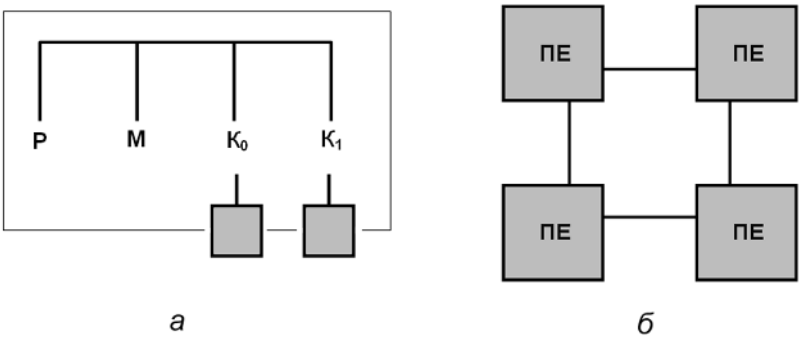


Рис. 4.8. Обчислювальна система з розподіленою пам’яттю:
а – процесорний елемент; *б* – об’єднання процесорних елементів

Цікавий варіант системи з розподіленою пам’яттю є модель розподіленої спільно використовуваної пам’яті (DSM, Distribute Shared Momory), відомої також і під іншою назвою архітектури з неоднорідним доступом до пам’яті і програмним забезпеченням когерентності (SC-NUMA, Software-Coherent Non-Uniform Memory Architecture). Ідея цієї моделі полягає в тому, що ОС, фізично будучи системою з розподіленою пам’яттю, завдяки операційній системі представляється користувачеві як система із загальною пам’яттю. Це означає, що операційна система пропонує користувачеві єдиний адресний простір, не дивлячись на те, що фактичне звернення до пам’яті «чужого» комп’ютера ОС як і раніше забезпечується шляхом обміну повідомленнями.

4.2.4 Мультипроцесорна когерентність кеш-пам’яті

Мультипроцесорна система з пам’яттю, що розділяється, складається з двох або більш незалежних процесорів, кожен з яких виконує або частину великої програми, або незалежну програму. Всі процесори звертаються до команд і даних, що зберігаються в загальній основній пам’яті. Оскільки пам’ять є спільним ресурсом, при зверненні до неї між процесорами виникає суперництво, в результаті цього середня затримка на доступ до пам’яті збільшується. Для скорочення такої затримки кожному процесору додається локальна кеш-пам’ять, яка, обслуговуючи локальні звернення до пам’яті, у

багатьох випадках запобігає необхідності доступу до спільно використовуваної основної пам’яті. У свою чергу, оснащення кожного процесора локальною кеш-пам’яттю приводить до так званої проблеми когерентності або забезпечення узгодженості кеш-пам’яті. Система є когерентною, якщо кожна операція читання за якою-небудь адресою, виконана будь-яким з процесорів, повертає значення, що занесене в ході останньої операції запису за цією адресою, незалежно від того, який з процесорів проводив запис останнім.

У простій формі проблему когерентності кеш-пам’яті можна пояснити таким чином (рис. 4.9). Нехай два процесори P_1 і P_2 пов’язано із загальною пам’яттю за допомогою шини. Спочатку обидва процесори читають змінну *x*. Копії блоків, що містять цю змінну, пересилаються з основної пам’яті в локальні кеші обох процесорів (рис.4.9, *а*). Далі процесор P_1 виконує операцію збільшення значення змінної *x* на одиницю. Оскільки копія змінної вже знаходиться в кеш-пам’яті даною процесора, відбудеться кеш-попадання і значення *x* буде змінено тільки в кеш-пам’яті 1. Якщо тепер процесор P_2 знов виконає операцію читання *x*, то також відбудеться кеш-попадання і P_2 прочитає «застаріле» значення *x*, що зберігається в його кеш-пам’яті (рис. 4.9, *б*).

Підтримка узгодженості вимагає, щоб при зміні елементу даних одним з процесорів відповідні зміни були проведені в кеш-пам’яті решти процесорів, де є копія зміненого елементу даних, а також в загальній пам’яті. Схожа проблема виникає, до речі, і в однопроцесорних системах, де присутньо декілька рівнів кеш-пам’яті. Тут потрібно погоджувати вміст кешів різних рівнів.

У вирішенні проблеми когерентності виділяються два підходи програмний і апаратний. У деяких системах застосовують стратегії, суміщаючі обидва підходи.

Програмні прийоми вирішення проблеми когерентності дозволяють обійтися без додаткового устаткування або звести його до мінімуму. Задача

покладається на компілятор і операційну систему. Привабливість такого підходу в можливості усунення некоректності ще до етапу виконання програми, проте ухвалені компілятором рішення можуть в цілому негативно позначитися на ефективності кеш-пам’яті.

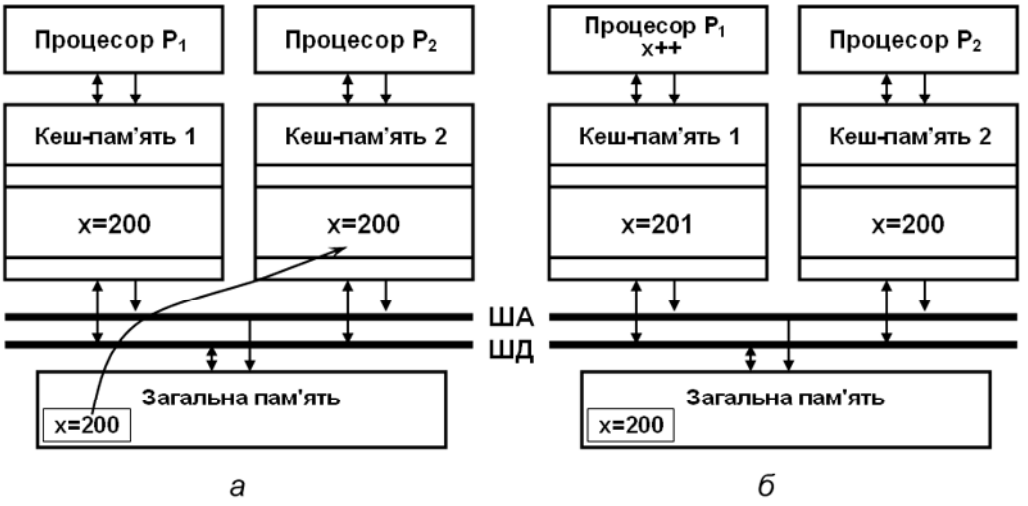


Рис. 4.9. Ілюстрація проблеми когерентності пам’яті:
а — вміст пам’яті до зміни значення *x*; *б* — після зміни.

Компілятор аналізує програмний код, визначає ті спільно використовувані дані, які можуть стати причиною некогерентності, і позначає їх. В процесі виконання програми операційна система або відповідна апаратура запобігають кешуванню (занесення в кеш-пам’ять) помічених даних, і надалі для доступ до них, як при читанні, так і при записі, доводиться звертатися до «повільної» основної пам’яті. Враховуючи, що некогерентність виникає тільки в результаті операцій запису, що відбуваються значно рідше, ніж читання, розглянутий прийом слід визнати недостатньо вдалим.

Ефективнішими представляються способи, де в ході аналізу програми визначаються безпечні періоди використання загальних змінних і так звані критичні періоди, де може виявитися некогерентність. Потім компілятор вставляє в генерований код інструкції, що дозволяють забезпечити когерентність кеш-пам’яті саме в такі критичні періоди.

Більшість із апаратних способів боротьби з некогерентністю орієнтовані на динамічне (в процесі обчислень) розпізнавання і усунення

неузгодженості копій спільно використовуваних даних за допомогою спеціальної апаратури. Апаратні методи забезпечують вищу продуктивність, оскільки витрати, пов’язані з некогерентністю, мають місце тільки при виникненні ситуації некогерентності. Крім того, непрограмний підхід прозорий для програміста і користувача. Апаратні механізми подолання проблеми когерентності прийнято називати протоколами когерентності кеш-пам’яті.

Як відомо, для забезпечення ідентичності копій даних в кеші і основної пам’яті в однопроцесорних системах застосовується одна з двох стратегій: наскрізний запис (write through) або зворотний запис (write back). При наскрізному записі нова інформація одночасно заноситься як в кеш, так і в основну пам’ять. При зворотному записі всі зміни проводяться тільки в кеш-пам’яті, а оновлення вмісту основної пам’яті відбувається лише при видаленні блоку з кеш-пам’яті шляхом пересилки блоку, що видаляється, у відповідне місце основної пам’яті. У разі мультипроцесорної системи, коли копії спільно використовуваних даних можуть знаходитися відразу в декількох кешах, необхідно забезпечити когерентність всіх копій. Ні наснаскрізний, ні зворотний запис не передбачають такої ситуації, і для її дозволу спираються на інші прийоми, а саме: запис з анулюванням (write invalidate) і запис з оновленням (write update). Останній прийом відомий також під назвою запису з трансляцією (write broadcast).

У варіанті запису з анулюванням, якщо який-небудь процесор проводить зміни в одному з блоків своєї кеш-пам’яті, всі наявні копії цього блоку в інших локальних кешах анулюються, тобто позначаються як недостовірні. Для цього біт достовірності зміненого блоку у всіх інших кешах встановлюється в 0.

Контрольні питання до розділу 4

1. Сформулюйте основні принципи комп’ютерного введення-виведення.
2. Назвіть основні функції СВВ і способи їх реалізації
3. Поясніть суть переривань.
4. Поясніть суть призупинок.
5. Загальні принципи організації пам’яті в комп’ютерних системах. Особливості організації пам’яті із чергуванням адрес.
6. Класифікація моделей архітектур пам’яті обчислювальних комп’ютерних систем.
7. Моделі архітектур із загальною пам’яттю: UMA, NUMA. Переваги і недоліки.
8. Моделі архітектур із загальною пам’яттю: COMA, CC-NUMA, NCC-NUMA. Переваги і недоліки.
9. Моделі архітектур з розподіленою пам’яттю. Переваги і недоліки.
10. Мультипроцесорна когерентність кеш-пам’яті та програмні методи її вирішення.
11. Апаратні способи вирішення проблеми когерентності кеш-пам’яті в комп’ютерних системах. Переваги і недоліки.

РОЗДІЛ 5. НАДІЙНІСТЬ ТА ЕКСПЛУАТАЦІЯ КС

Тема 5.1. Надійність КС

5.1.1 Основи надійності КС

Паралельні обчислювальні системи будують зокрема для того, щоб досягти унікально високих значень показників надійності, продуктивності і інших показників якості функціонування, наприклад продуктивності на одиницю вартості обслуговування.

Як показники надійності зазвичай використовують середній час між відмовами *T*, або інтенсивність відмов, а також коефіцієнт готовності *Кг*, що визначається як вірогідність того, що відновлювана (ремонтована) ОС буде працездатна в будь-який довільно вибраний момент часу в стаціонарному режимі функціонування:

$$K_г = T/(T+t), \qquad (5.1)$$

де *t* — середній час відновлення.

Також часто використовується поняття коефіцієнту простою, значення якого визначається як $K_п = 1 - K_г$.

Під відмовою ОС розуміється стійке неправильне функціонування апаратно-програмних засобів, унаслідок виникнення несправності або прояву невиявленої раніше помилки в програмно-апаратних засобах. Під збоєм ОС розуміється разове, випадкове неправильне функціонування, обумовлене виникаючою несправністю, збуреннями зовнішнього середовища, а також реалізацією властивих електронній апаратурі «неправильних» спрацьовувань відповідно до функції розподілу вірогідності таких подій.

Підвищення показників надійності досягається введенням апаратної, інформаційної або тимчасової надлишковості. Розподіл надлишковості по компонентах ОС складає суть роботи по забезпеченню надійності. Необхідно виділити компоненти ОС, функціонування яких визначає працездатність, і

ввести альтернативні варіанти функціонування цих компонент при відмові частини складових їх апаратно-програмних засобів.

При цьому задається набір відмов, які не приводять до втрати працездатності, і, відповідно, визначається сукупність відмов, що обумовлює втрату працездатності. У цьому сенсі абсолютно відмовостійких ОС не існує, можна побудувати ОС, що зберігає працездатність при передбаченій кратності відмов кожного типу використовуваних апаратно-програмних компонент.

Одна з основних властивостей паралельних систем — можливість забезпечення необхідних показників надійності при будь-яких видах дій, включаючи знищення комп'ютерних центрів в результаті пожеж і інших катастроф. Це досягається за рахунок просторового рознесення на значні відстані компонент ОС. Забезпечувана при цьому постійна доступність дозволяє довіряти таким ОС життєво важливі функції управління.

Підвищення показників надійності окремого комп'ютера, що як правило виконує функції сервера, досягається блокуванням дій компонентів, що відмовили, і їх оперативною заміною:

- відмова процесора — використання багатопроцесорних конфігурацій;
- відмова пам'яті — використання кодів з виявленням і виправленням помилок;
- відмова дисків — застосування RAID технологій, а також гарячої заміни (без переривання функціонування комп'ютера) дисків, що відмовили, і контроллерів;
- відмова живлення і охолодження — використання резервних блоків живлення і блоків вентиляторів, джерел безперебійного живлення, а також гарячу їх заміну.

Об'єднання таких комп'ютерів у ОС створює наступний рівень забезпечення надійності — взаємний контроль комп'ютерами один одного з метою виявлення помилкових дій персоналу, адекватної реакції на зміну

зовнішнього середовища і так далі.

Коефіцієнт готовності окремого вузла (комп'ютера) без застосування засобів блокування відмов зазвичай становить 0.99. Це складає близько 4 днів в році. Об'єднання вузлів у систему і застосування вищеперелічених заходів дозволяє довести *Кг* до 0.9999, що зводить простій ОС до декількох хвилин в рік.

5.1.2 Методи підвищення надійності контролю КС

Методи підвищення надійності можна розділити на структурні та інформаційні.

Структурні методи підвищення надійності. Абсолютною надійності технічних пристроїв домогтися принципово неможливо, а максимально підвищити показники їх надійності реально, і це є найважливішим науковим і технічним завданням. Підвищення рівня надійності систем досягається, перш за все, усуненням причин, що викликають у ній відмови, тобто зведенням до мінімуму конструкторських, технологічних та експлуатаційних помилок.

Значного підвищення надійності КС досягають створенням нових елементів. Так, застосування інтегральних схем для побудови КС призвело до значного підвищення надійності апаратури третього і четвертого поколінь.

Однак підвищенням надійності елементів не вдається повністю вирішити проблему побудови надійних КС, що викликано значним випередженням зростання складності розроблених нових КС, великими витратами при отриманні елементів високої надійності, а також існуванням елементів, надійність яких досить низька і важко піддається підвищенню. Тому один із шляхів підвищення надійності РЕА - введення схемної надлишковості.

Інформаційні методи підвищення надійності. Основне застосування інформаційні методи знаходять в обчислювальній техніці. Реалізуються вони

у вигляді коригувальних кодів. Призначення цих кодів складається в тому, щоб виявляти і виправляти помилки в КС без переривання їхньої роботи.

Коригувальні коди передбачають введення в вироби деякої надлишковості. Розрізняють тимчасову і просторову надлишковість. Тимчасова надлишковість характеризується неодноразовим рішенням завдання. Отримані результати порівнюються, і якщо вони збігаються, то робиться висновок, що задача вирішена правильно. Тимчасова надмірність вводиться в РЕА програмним шляхом.

Просторова надлишковість характеризується подовженням кодів чисел, в які вводять додатково контрольні розряди. Суть виявлення та виправлення помилок за допомогою коригувальних кодів складається в наступному. У множині А вихідних слів пристрою виділяють підмножину В дозволених кодових слів (тобто $B \in A$). Ці слова можуть з'явитися лише в тому випадку, якщо всі арифметичні і логічні операції, що виконуються КС, здійснюються правильно. Тоді очевидно, що підмножина A - B = C буде характеризувати заборонені кодові слова. Останні мають місце тільки при наявності помилок.

Далі всі слова на виході пристрою аналізують. Наприклад, якщо слово b_I відноситься до підмножини дозволених кодових слів (тобто $b \in B$), то це означає, що процес йде нормально; слово b_I вважають правильним і його можна декодувати.

Якщо на виході пристрою з'являється заборонене кодове слово c_i ($c_i \in C$), то це свідчить про наявність помилки, і вона фіксується.

Для усунення виявлених таким чином помилок усі заборонені кодові слова розбиваються на групи. Кожній такій групі ставиться у відповідність тільки одне дозволене кодове слово. При декодуванні заборонені кодові слова c_i автоматично замінюються дозволеними кодовими словами з тієї групи, до якої належить c_i. Таким чином, коригувальні коди в стані не тільки виявляти помилки, але й усувати їх.

Резервування - спосіб підвищення надійності апаратури, який полягає у дублюванні систем в цілому або окремих її модулів або елементів. Резервування передбачає включення в схему пристрою додаткових елементів, які дозволяють компенсувати відмови окремих частин пристроїв та забезпечити його надійну роботу. Але резервування ефективно тільки в тому випадку, коли несправність є статистично незалежними.

Розрізняють загальне резервування (резервується об’єкт в цілому) та роздільне резервування - резервування, при якому резервуються окремі елементи об’єкта або їх групи.

Структурне резервування - метод підвищення надійності об’єкта, який передбачає використання надлишкових елементів, що входять у фізичну структуру об’єкта.

Тимчасове резервування - метод підвищення надійності об’єкта, який передбачає використання надлишкового часу, виділеного для виконання завдань.

Інформаційне резервування - метод підвищення надійності об’єкта, який передбачає використання надлишкової інформації понад мінімально необхідної для виконання завдань.

Функціональне резервування - метод підвищення надійності об’єкта, який передбачає використання здатності елементів виконувати додатні функції замість основних або поряд з ними.

Навантажувальне резервування - метод підвищення надійності об’єкта, який передбачає використання здатності його елементів сприймати додаткові навантаження понад номінальних.

Основний елемент - елемент основної фізичної структури об’єкта, мінімально необхідної для нормального виконання об’єктом його завдань.

Резервний елемент - елемент, призначений для забезпечення працездатності об’єкта в разі відмови основного елемента.

Дублювання - резервування, при якому одному основному елементу надається один резервний

Кратність резервування-відношення числа резервних елементів до числа резервних елементів об’єкта.

Розрізняють такі види резервування: постійне (резервні елементи включені разом з основним і функціонують в тих же режимах); резервування заміщенням (виявлення відмовив елемента і заміна його резервним); ковзне резервування (будь-який резервний елемент може заміщати будь-який елемент, що відмовив).

Якщо Рс(і) - ймовірність безвідмовної роботи системи, то установка і включення паралельно декількох таких же систем призводить до збільшення результуючої ймовірності безвідмовної роботи резервованих системи Р (і), яку можна визначити з виразу:

$$P\ (i) = 1 - [1-Pc\ (i)]^{\tau+1}, \qquad (5.2)$$

де τ - число резервних систем, що включені паралельно основній.

Так, наприклад, при ймовірності безвідмовної роботи модуля 0,7 включення одного резервного модуля підвищить ймовірність безвідмовної роботи до 0,91, а двох - до 0,973.

Постійне резервування в КС виконують за такою схемою: вхідні сигнали надходять на *n* логічних схем, причому $n > k$, де *k* - число логічних схем в нерезервованих схемою. Вихідні сигнали всіх *n* логічних схем далі подають на вирішальний елемент, який згідно функції рішення з цих сигналів визначає значення вихідних сигналів всієї схеми. Функція рішення - правило відображення вхідних станів вирішального елементу на безліч його вихідних станів.

Найпростіший і найпоширеніший вид функції рішення - «закон більшості» або мажоритарний закон. Вирішальний елемент зазвичай називають мажоритарним елементом. Робота мажоритарного елемента полягає в наступному: на входи елемента надходять двійкові сигнали від непарної кількості ідентичних елементів; вихідний сигнал елемента приймає значення, рівне значенню, яке приймає більшість вхідних сигналів. Найбільш

широко використовують мажоритарні елементи, що працюють за законом «2 з 3». У цих елементах значення вихідного сигналу дорівнює значенню двох однакових вхідних сигналів.

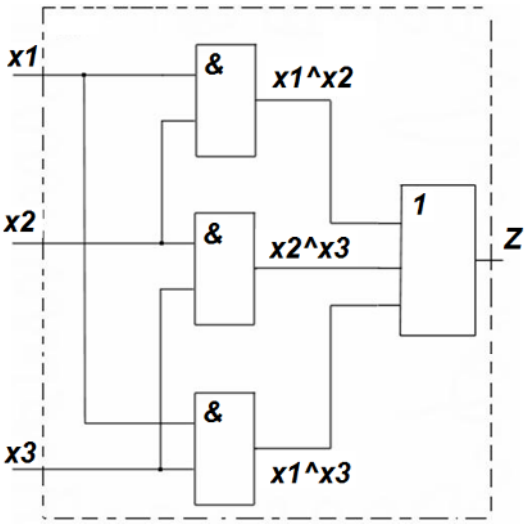


Рис. 5.1 Схема мажоритарного елемента «2 з 3»

Крім того, відомі мажоритарні елементи, що працюють за законом «3 з 5», «4 з 7» і т. д. Схема мажоритарного елемента, що працює за законом «2 з 3» і побудованого з логічних елементів І та АБО, наведена на рис. 5.1 та характеризується виразом: $Z = x_1 x_2 + x_2 x_3 + x_1 x_3$.

За способом включення резервних елементів функціональних пристроїв розрізняють три види резервування: постійне, заміщенням і ковзне.

При постійному резервуванні припускають, що будь-який елемент, що відмовив, або вузол не впливає на вихідні сигнали і тому його прямого виявлення не проводиться.

Постійне резервування найбільш поширене в не відновлюваних пристроях. Крім того, воно є єдино можливим в пристроях, де неприпустимий навіть короткочасна перерва в роботі.

Постійне резервування вводиться або за допомогою вирішувального блоку, або у вигляді однотипних елементів або блоків, включених послідовно, паралельно або, наприклад, згідно із законами *n*-кратної логіки.

Як вирішувальний блок використовується мажоритарні елементи з постійними або змінними кодуючими вагами - пристрої декодування та схеми з логічних елементів І, АБО, НЕ.

Резервування заміщенням передбачає виявлення елемента або вузла, що відмовив, та підключення справного. Заміщення може відбуватися або автоматично, або вручну.

Резервування заміщенням має такі переваги. Для багатьох схем при включенні резервного обладнання не потрібно додатково регулювати вихідні параметри, внаслідок того, що електричні режими в схемі не змінюються. Резервна апаратура до моменту включення в роботу знеструмлена, що підвищує загальну надійність системи за рахунок збереження ресурсу електронних пристроїв. Є можливість використання одного резервного елемента на кілька робітників.

Внаслідок складності апаратури для автоматичного включення резерву резервування заміщенням доцільно застосовувати до великих блоків і окремих функціональних частин КС.

При ковзному резервування будь-який резервний елемент може заміщати будь-який основний елемент. Для здійснення цього резервування необхідно мати пристрій, який автоматично знаходить несправний елемент і підключає замість нього резервний. Перевага такого резервування в тому, що при ідеальному автоматичному пристрої найбільший виграш буде в надійності в порівнянні з іншими методами резервування. Проте здійснення ковзного резервування можливе лише при однотипності елементів.

Тема 5.2. Експлуатація КС

5.2.1 Основи технічного обслуговування КС

Технічне обслуговування та експлуатація КС (профілактичне обслуговування) – це система попереджувальних мероприємств, межою яких є зниження ймовірності виникнення відмов. До них відносяться технічні огляди, вимір параметрів, прогон контрольних тестів, регуліровки і налаштування, заміна елементів, що находяться в близькому до відмови стані, чистка і промивка контактів, мастильні роботи, відновлення захисного покриття, загальна чистка від забруднення.

Технічне обслуговування (ТО) передбачає дві цілі:

– вивести і відремонтувати накопичені відмови, які не відновлювались в процесі роботи ЕОМ;
– упередження можливих відмов.

Існує три основні системи ТО:

– автономна;
– централізована;
– змішана.

5.2.2 Автономна система ТО

Передбачає виконання всіх робіт по обслуговуванню й ремонту засобів обчислювальної техніки (ЗОТ) персоналом, експлуатуючим ЕОМ. Ця система ТО являється першим ступенем організації систем обслуговування, сама розповсюджена в данний час, але маюча ряд суттєвих недоліків:

а) великий об’ем необхідних запасних блоків, інструменту і приладів (ЗІП);

б) потрібна висока кваліфікація обслуговуючого персоналу широкого профілю, або декілька спеціалістів різного напрямку. Коефіціент зайнятості спеціалістів дуже низький – не вище 0,3;

в) потрібен повний комплект стендів і вимірчих приладів, які використовуються неефективно;

г) при ремонті блоків і вузлів відновлення виконується не повністю. Рівень якості відновлення блоків нижче начального, так як ряд складних технологічних процесів неможливо виконати в лабораторних умовах.

Всі ці недоліки посилюються, якщо ЗОТ побудовані з неуніфіцированих блоків і кожного споживача мала кількість ЕОМ.

5.2.3 Централізована система ТО

Передбачає наявність центра технічного обслуговування із структурою, що наведена на рис. 5.2.

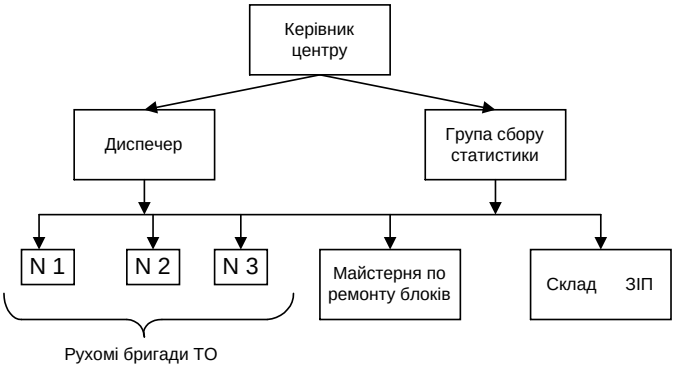


Рис. 5.2 Структура центру ТО

Центри ТО повинні створювать по регіональному принципу. Структурна схема системи централізованого обслуговування наведена на рис. 5.3.

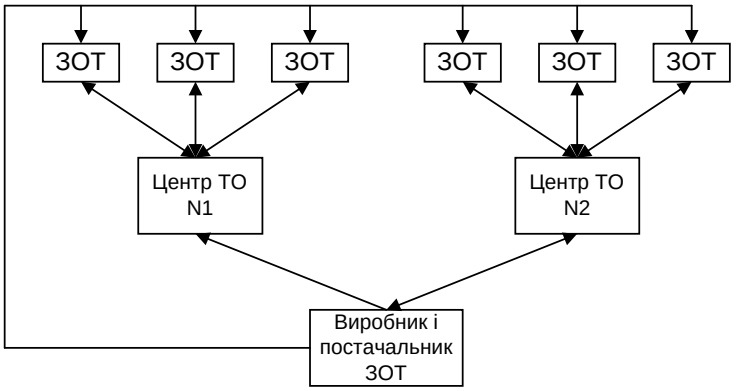


Рис. 5.3.Централізована система ТО

При централізованому обслуговуванні весь обслуговуючий, ремонтний персонал і ЗІП знаходиться в центрі технічного обслуговування (ЦТО). ЦТО має ряд рухомих бригад і транспортні засоби для виконання планових ТО і аварійних ремонтів. При централізованому ТО на місцях експлуатації не передбачається виконувати ніяких ремонтно-профілактичних робіт силами експлуатуючого персоналу. Всі ці роботи повинні виконувати співробітники ЦТО.Планове і профілактичне обслуговування виконується рухомими бригадами по спеціальних графіках, розроблених для кожного обчислювального засобу (комплексу) з врахуванням його структурних особливостей, місця розташування, інтенсивності експлуатації, важливості об’єкту автоматизації та іншого. По цих графіках рухомі бригади об’їжджають всі ЗОТ і виконують всі необхідні профілактичні роботи, передбачені інструкцією по експлуатації ЗОТ. При відмовах ЗОТ під час експлуатації викликається рухома бригада для екстреного (аварійного) відновлення. Такий виклик робиться через диспетчера ЦТО і відновлення виконується поза графіками в аварійному порядку.

Перевагами цієї системи являється:

а) високий технічний рівень персонала ЦТО;

б) високий рівень ймовірності достатності ЗІП;

в) можливість мати унікальне обладнання для виконання відновлення;

г) високий коефіцієнт навантаження персоналу і обладнання;

д) швидкий оберт ЗІП;

е) кваліфіковане і якісне виконання ТО і відновлень;

ж) збір достовірної статистики про надійність ЗОТ.

До недоліків відноситься невисока оперативність виконання раптових відмов і ТО.

5.2.4 Змішана система ТО

Об’єднує в собі переваги автономної і централізованої систем. Вона передбачає наявність обмеження можливостей по обслуговуванню і відновленню нескладних відмов ЗОТ на місці експлуатації силами місцевого експлуатуючого персоналу. Для цього на місцях експлуатації ЕОМ знаходиться ЗІП з невеликою ймовірністю достатності, але включаючий найбільше ненадійні елементи. Складне технічне обслуговування і складні ремонти виконує персона ЦТО. Планове обслуговування виконується по графіку так же, як при централізованому обслуговуванні персоналом ЦТО. Він же виконує обмін відмовивших елементів і блоків і поповнює оперативний ЗІП.

При довільній з наведених систем ТО для їх створення і ефективного функціонування необхідно вирішувати три складні в методичному і математичному плані задачі:

а) розрахунок кількості і номенклатури ЗІП, необхідного на місцях експлуатації і в ЦТО;

б) розрахунок оптимальної кількості і кваліфікації обслуговуючого персоналу;

в) розрахунок план-графіку проведення ТО при багаторівневому обслуговуванні і додержанні високого коефіцієнту використання (K_B).

Тема 5.3. Діагостика КС

5.3.1 Основи діагностики КС

Забезпечення відмовостійкої КС включає вирішення наступних проблем:

1) виявлення збоїв або відмов програмно-апаратних засобів;

2) діагностування збоїв або відмов і усунення їх впливів;

3) відновлення працездатності шляхом використання для продовження функціонування визнаних працездатними процесорів і прикладних програм, що вони виконують, або шляхом перезапуску загальносистемних і прикладних програм на реконфігурованій ОС.

Кожна з вищеперелічених проблем може вирішуватися різними методами, причому програмно-апаратні витрати на їх рішення залежать від інтервалу часу, допустимого за умовами експлуатації ОС, від моменту збою або відмови до моменту відновлення працездатності.

5.3.2 Методи виявлення збоїв чи відмов у КС

Збої або відмови виявляються, зокрема, шляхом:

- використання кодів з виявленням і виправленням помилок;
- порівняння в кожному такті результатів виконання однієї і тієї ж програми на двох і більше процесорах і вироблення сигналу зупинки у разі неспівпадання результатів;
- виконань однієї і або функціонально еквівалентних програм (запрограмованих по різних методах і, можливо, на різних мовах програмування) на різних процесорах і порівняння результатів в передбачених точках синхронізації, наприклад при зверненні до зовнішніх пристроїв;
- перевірки працездатності вузлів і інших програмно-апаратних компонент за допомогою запуску тестових процедур, що активізуються або

шляхом обміну між вузлами спеціальними повідомленнями (heartbeat), або перериваннями від таймерів.

Перші три методи суміщають виконання обчислень і виявлення збоїв і відмов в одній виконуваній програмі. Четвертий метод використовує тестові програми, відокремлені від виконуваних програм користувачів. Відповідно можливі альтернативи використання цих методів визначаються тим, що важливіше: 1) витратити більше ресурсів і повільніше, ніж максимально можливо, виконувати програму, виявляючи збої і відмови, або 2) виконувати програму з максимально можливою швидкістю і проводити через деякі проміжки часу тестування на предмет виявлення відмов. Вибір між цими альтернативами слід проводити з урахуванням того, що ресурси, що витрачаються на виявлення збоїв і відмов, можуть бути використані для прискорення паралельного виконання програми, що у свою чергу, веде до утворення часової надлишковості. Остання може бути також використана для підвищення відмовостійкої. Наприклад, часова надлишковість дозволяє повторювати обчислення і порівнювати результати, що виявляє збої при виконання програми.

На рівні окремого вузла для виявлення збоїв і відмов використовується декілька мікропроцесорів, що виконують синхронно команди програми з порівнянням результатів в кожному такті. На рис. 5.4, *а* представлені структури таких систем, що складаються з трьох і двох процесорів відповідно. Можливі і інші структури, наприклад, Tandem використовує вузли, що складаються з двох пар процесорів з попарним порівнянням результатів і використанням того результату, який є однаковим у обох процесорів однієї пари.

У разі неспівпадання результатів, шляхом реконфігурації виключається вплив несправного процесора. Після реконфігурації продовжується виконання прикладних програм на визнаних справними процесорах. У цих системах виконувані прикладні програми не сповіщають про збої і відмови, що відбулися.

Виявлення збоїв і відмов на рівні групи вузлів організовується шляхом виконання програми на кожному вузлі групи і порівняння результатів, шляхом пересилки їх між вузлами. В цьому випадку потрібна спеціальна організація програми як сукупності копій, що виконуються на різних процесорах. Вузли ухвалюють рішення про збій, що відбувся, або відмову і реконфігурують виконувану програму, перерозподіляючи копії по працездатних машинах. На рис. 5.4, *б* показано фрагмент програми з двох блоків a і b, який перетворюється на фрагмент з шести блоків: трьох копій a_i блоку a і трьох копій b_i, блоку b, i=l, 2, 3.

Можна говорити про статичну надмірність у разі виявлення відмов на рівні окремої ЕОМ і динамічної надлишковості у разі виявлення на рівні групи ЕОМ. Взагалі кажучи, збій або відмова схеми голосування у вузлі із статичною надлишковістю приводить до неправильного функціонування ОС в цілому. Тому у відповідальних застосуваннях статична надмірність повинна доповнюватися динамічною, при використанні якої можуть бути застосовані спеціальні алгоритми голосування, складність яких не регламентується потужністю вузла. У цьому сенсі динамічна надлишковість більш притаманна для масово паралельних ОС, оскільки із збільшенням числа вузлів збільшуються можливості реконфігурації.

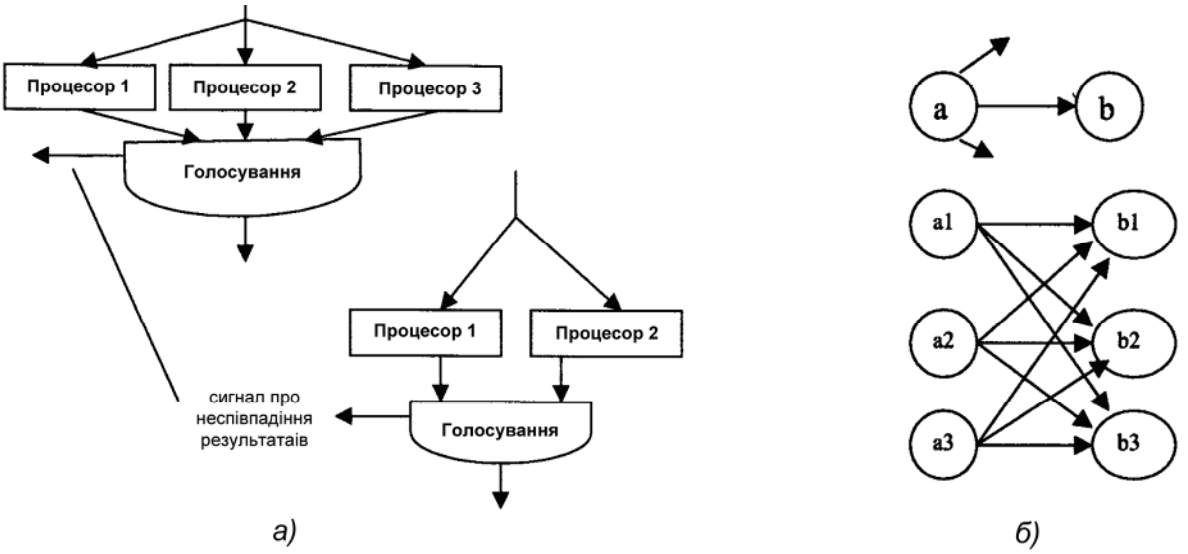


Рис. 5.4 Методи виявлення збоїв і відмов

При динамічній надлишковості можливе використання специфіки виконуваних прикладних програм для зменшення складності реалізації голосування. Крім того, кожна програма, що має власну схему голосування, не залежить від інших програм і не впливає на інші додатки у разі своєї відмови.

До недоліків динамічної надлишковості можна віднести велике число виконуваних копій і значне зростання числа міжмодульних обмінів. При використанні n копій кількість обмінів зростає в n^2 разів.

Вирішення проблеми генерування вузлами, узгоджених рішень за результатами виконання копій тестуючих програмних блоків представляє складну проблему.

Як ілюстрація сказаного розглянемо наступний приклад. Нехай є фрагмент програми, хід виконання якої наведено на рис. 5.5.

Вважатимемо, що оператори, що виконуються в обчислювальних модулях B_i, i=l, 2, 3, отримують по різних лініях копії a_i одного і того ж значення. Розглянемо процедуру узгодження модулями дійсного значення змінної по отриманих значеннях її копій. Припустимо, що при передачі копія може бути спотворена і вузли можуть мати несправності, які при обробці як правильних, так і спотворених копій можуть викликати видачу вузлом довільного висновку про правильність оброблюваної копії. При такому припущенні про прояви несправностей, обмін копіями між вузлами і ухваленням рішення по більшості співпадаючих копій у разі трьох ВМ не дає правильного результату, що ілюструється на рис. 5.5.б. Нехай вузол В1 не справний, а копія a_3 спотворена. На рис 5.5.б. б спотворені копії відмічені чорними мітками, а не спотворені — світлими. У наведеному прикладі модуль В2 отримає дві світлі мітки і генерує одне представлення про правильну копію, а модуль В3 отримає дві чорні мітки і визнає правильною спотворену копію.

Для досягнення правильного результату в голосування щодо виявлення одного несправного вузла необхідне використання чотирьох копій, як

показано на рис. 5.5.*в*. Це випливає з умов вирішення проблеми «візантійських генералів». Показано, що правильне голосування можливетільки тоді, якщо більше, ніж дві третини вузлів справні в припущенні про одну допустиму несправність.

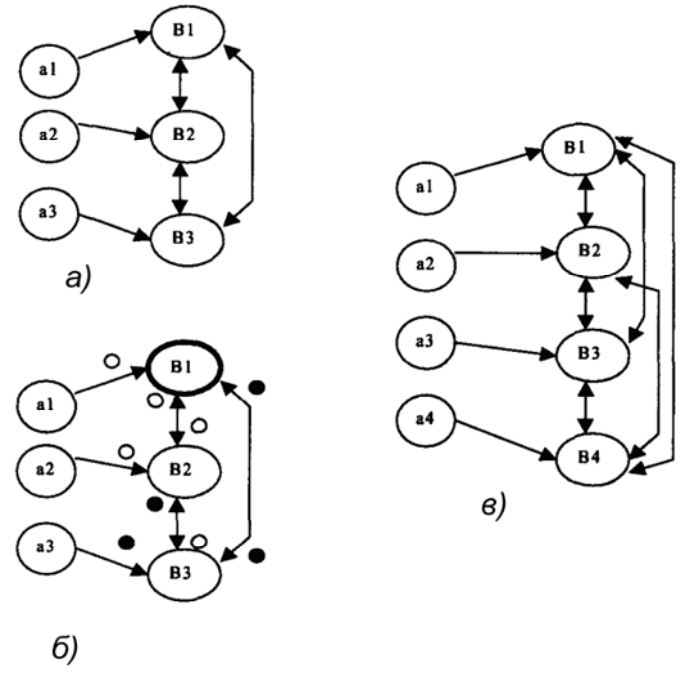


Рис. 5.5 Ілюстрація проблеми виявлення несправного вузла

5.3.3 Методи тестового виявлення відмов

Методи тестового виявлення відмов базуються на збереженні в зовнішній пам’яті з підвищеною відмовостійкою або в сусідніх вузлах так званих контрольних точок (КТ) кожного вузла, які задають стан його програми. Виконання програми полягає у виконанні наступної послідовності дій:

1) запуск програми з поточної контрольної точки (перша КТ — початковий стан програми і даних) і виконання обчислень до наступної контрольної точки;

2) діагностичне тестування ОС: при виявленні відмови - формування працездатної конфігурації програмно-апаратних ресурсів і повторення

обчислень з попередньою КТ, за відсутності відмови - збереження нової КТ як поточної і запуск програми з поточної КТ.

При виконанні паралельних програм між вузлами передаються повідомлення, що обумовлює необхідність спеціальним чином підходити до вибору положення контрольних точок у вузлах. Для пояснення суті цієї проблеми розглянемо хід обчислень в трьох вузлах, що наведено на рис.3.3.

Отже, на рис. 5.6, *а* Вузол_а має КТ al, a2, a3, Вузол_b- b1, b2, b3, Вузол_с — cl, c2, c3. Двонаправлені стрілки відображають передачу повідомлень між вузлами, причому напрям передачі для нашого випадку не має значення. Нехай Вузол_а після збереження КТ а3 продовжив обчислення, обмінявся повідомленням з Вузлом_b і досяг наступної контрольної точки. Проте, нехай при цьому буде виявлена відмова Вузла_а. В цьому випадку необхідно відновити обчислення у Вузлі_а з контрольної точки а3, але після колишнього старту з цією КТ відбувся обмін повідомленням з Вузлом_b. Тому для коректного виконання паралельної програми необхідно повернутися у Вузіл_b до КТ b3, але далі для коректного виконання доведеться повернутися у всіх вузлах аж до початку обчислень. Це так званий «ефект доміно».

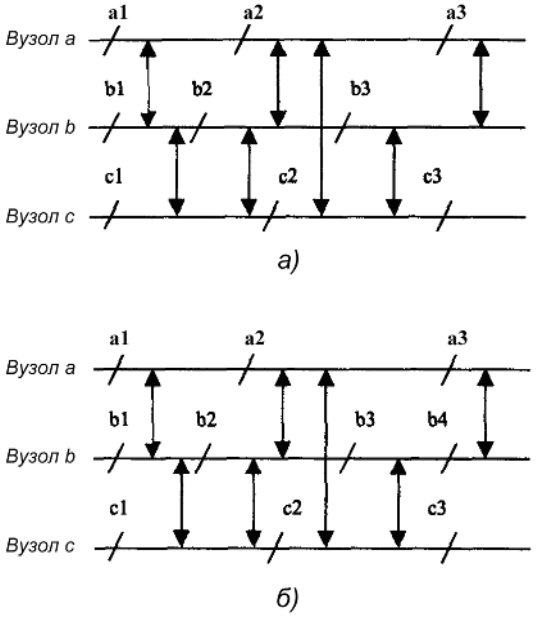


Рис.5.6 Методи тестового виявлення відмов

Щоб підвищити ефективність паралельних обчислень необхідно виключити можливість множинних відкатів, що вимагає збереження великої кількості контрольних точок і великого об’єму повторних обчислень. Це може бути досягнуто шляхом використанням системних контрольних точок, що зберігають стан всіх вузлів після взаємодії між ними. Приклад системної КТ, що складається з a3, b4, c3 показаний на рис.5.6, *б*.

Контрольні питання до розділу 5

1. Назвіть основні показники відмовостійності КС.
2. Назвіть способи підвищення надійності КС
3. Які є види резервування?
4. Які цілі технічного обслуговування КС?
5. Назвіть основні системи ТО?
6. Поріняйте централізовану і змішану схему ТО?
7. Що таке відмова КС?
8. Що таке збій КС?
9. Назвіть методи виявлення збоїв та відмов.
10. Які є основні алгоритми виявлення збоїв?
11. Що таке принцип «доміно»?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. – СПб.: БХВ-Питербург, 2004. – 608с.

2. Гергель В.П. Теория и практика параллельных вычислений. Учебное пособие. - М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007.

3. Головки Б.А. Параллельные вычислительные системы. – М.: Наука, 1980 – 520с.

4. Глушаков С.В., А.В. Коваль, С.В. Смирнов. Язык программирования С++. – Харьков: Фолио. – 2002.

4. Корнеев В.В. Параллельные вычислительные системы. – М.: «Нолидж», 1999., - 320с.

5. Корнеев В. В., Современные микропроцессоры, издательство М.: “Нолидж”, БХВ , 2003

6. Мельник А.О. Архітектура комп’ютера.Наукове видання. – Луцьк.: Волинська обласна друкарня, 2008. – 470 с.

7. Немнюгин С., Стесик О. Параллельное программирование для многопроцессорных вычислительных систем СПб:, БХВ-Петербург, 2002

8. Нечаев Ю.И. Искусственный интеллект: концепции и приложения. СПб: Изд.центр СПбГМТУ, 2002.

9. Организация ЭВМ. 5-е изд../ К. Хамахер, З. Вранешич.,С. Заки. . – СПб.: Питер; Киев: изд. группа BXV, 2003. – 848с.

10. Смирнов А.Д. Архитектура вычислительных систем: Учебное пособие для вузов. - М: Наука, 1990.

11. Специализированные процессоры для высокопроизводительной обработки данных. / О.Л. Бадман, Н.Н. Миренков и др. Новосибирск: Наука, 1988.

12. СуперЭВМ. Аппаратная и программная организация. Под ред. С. Фернбаха. - М.: Радио и связь, 1991.

13. Таненбаум, Э.Архитектура компьютера. СПб, Из-во «Питер», 2002.

14. Тарасенко В.П. Надійність комп’ютерних систем / В.П. Тарасенко, А.Ю.

Маламан, Ю.П. Черніченко, В.І. Корнійчук. – К., 2007. – 256 с.

15. Хамахер К., Вранешич З., Заки С., Организация ЭВМ. СПб, Из-во «Питер», 2003.

16. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб.: Питер, 2004. – 668с.